NASA-TM-79111 19790012581

NASA Technical Memorandum 79111

LEWIS HYBRID COMPUTING SYSTEM - USERS MANUAL

William M. Bruton and David S Cwynar
Lewis Research Center
Cleveland, Ohio

April 1979

1 Report No	2 Government Accession No	3 Recipient's Catalog No
NASA TM-79111		

4 Title and Subtitle	5 Report Date
LEWIS HYBRID COMPUTING SYSTEM - USERS MANUAL	April 1979
	6 Performing Organization Code

7 Author(s)	8 Performing Organization Report No
William M Bruton and David S Cwynar	E-9938
	10 Work Unit No

9 Performing Organization Name and Address	11 Contract or Grant No
National Aeronautics and Space Administration Lewis Research Center Cleveland, Ohio 44135	

12 Sponsoring Agency Name and Address	13 Type of Report and Period Covered
National Aeronautics and Space Administration Washington, D C 20546	Technical Memorandum
	14 Sponsoring Agency Code

15 Supplementary Notes

16 Abstract

The Lewis Research Center's Hybrid Simulation Lab contains a collection of analog, digital, and hybrid (combined analog and digital) computing equipment suitable for the dynamic simulation and analysis of complex systems This report is intended as a guide to users of these computing systems The report describes the available equipment and outlines procedures for its use Particular attention is given to the operation of the PACER 100 digital processor System software to accomplish the usual digital tasks such as compiling, editing, etc and Lewis-developed special purpose software are described

17 Key Words (Suggested by Author(s))	18 Distribution Statement
Hybrid computer; Simulation; Operating system; Software, Processor; Function generation, Data processing; Interrupts	Unclassified - unlimited STAR Category 60

19 Security Classif (of this report)	20 Security Classif (of this page)	21 No of Pages	22 Price*
Unclassified	Unclassified		

* For sale by the National Technical Information Service Springfield Virginia 22161

CONTENTS

INTRODUCTION

The Lewis Research Center's Hybrid Simulation Lab is located in the 8X6 SWT Research and Control Building. The Simulation Lab contains a collection of analog, digital, and hybrid (combined analog and digital) computing equipment suitable for the dynamic simulation and analysis of complex systems. The principle use of this equipment is for the simulation of the steady-state and transient performance of airbreathing propulsion systems and their components. Because of the computing power available, simulations can be constructed in great detail and can frequently run in real-time. This allows the simulations to be used to develop, evaluate, and check out control modes and to predict propulsion system performance and stability in conjuction with major experimental programs. Simulation results can significantly affect the details of experimental programs, often reducing the required amount of testing.

There are two hybrid computing systems within the Simulation Lab. Each hybrid computing system consists of a digital processor, two analog processors, and a communications interface for control of and data exchange between the digital and analog processors. The Simulation Lab also contains a central trunking system, which allows communication between the two computing systems.

This report is intended as a guide to users of these Simulation Lab facilities, supplementing the numerous detailed manuals and operating instructions that are available to the users (references 1-4). While this report does describe both the analog and digital processors, it primarily deals with the use of the PACER 100 digital processor. This is because the PACER 100 is a relatively new addition to the Simulation Lab and it was felt that a general, introductory guide to its operation was needed. The report covers the structure of the PACER 100 system and describes the supporting software that is available. This includes system software to accomplish the usual digital tasks of source creation, editing, compiling, loading, and executing. Also discussed is software developed at Lewis to aid users of the system. This includes function generation routines and data collection and display programs.

A large part of this report is devoted to describing the capabilities of INFORM, which is a software package developed for the PACER 100. INFORM can provide the user with the capabilities for interactive data collection and display. This report outlines procedures for organizing user's programs for operation in a time-shared interrupt environment. An existing program called EXEC1, which allows the user to execute three different programs at three different priority levels, is described. One of these programs may be INFORM and it may be operated as a background task, without interrupting normal program execution, to provide interactive data collection and display capabilities. The use of EXEC1 and INFORM are described in detail in reference 5.

PACER HYBRID COMPUTING SYSTEM

There are two PACER 600 hybrid computing systems, manufactured by Electronics Associates, Inc. (EAI), in the LeRC Hybrid Simulation Lab. Each system consists of a PACER 100 digital processor, two PACER 681 (680) parallel analog processors, and a PACER 693 communications interface for control and data exchange between the digital and analog processors.

The digital processors are 32K, 16 bit computers. Peripherals available for each system are a dual disk system, a CRT terminal as a control console with a hard copier, a line printer, a card reader, and a paper tape reader/punch which is used primarily as a backup system for the disk.

The following sections describe the available equipment in the digital, analog and interface systems. Also described is the central trunking system which allows communication between the two hybrid computing systems. References 1-4 describe the digital, analog, and interface systems in detail.

PACER 100 DIGITAL SYSTEM

Total Memory	32768 words at 16 bits/word
Cycle Time	1.0 Microseconds
Moving Head Disk	Two platters for 2.2 million words total
Card Reader	300 Cards per minute
Line Printer	165 Characters per second
High Speed Paper Tape Reader	300 Characters per second
High Speed Paper Tape Punch	120 Characters per second
Tektronix 4010 CRT terminal	
Hard Copier for Tektronix 4010 Crt terminal	

	CONSOLES A1,B1,B2	CONSOLE A2
Integrator-summer	30	30
Track/store-summer	12	12
Zero limit-summer	24	12
SJ-INV (interface)	24	24
SJ-INV (Fixed DFG)	0	2
QSM-Inverter	60	48
QSM-HG amplifier	30	24
VDFG-Inverter	0	6
Total amplifiers	180	158
SS Pots (2-terminal)	64	96
SS Pots (3-terminal)	16	24
HS Pots (2-terminal	12	12
Dig. Coef. Attenuators (DCA's)	40	0
Total attenuators	132	132
Multipliers	30	24
DCFG's (Digital)	8	0
VDFG's (Analog)	0	6
Fixed DFG's (Analog)	0	2
Variable Limiter Networks	12	12
Comparators	24	24
Function Relays	24	24
D/A switches	24	24
General-purpose registers	6	6
"AND" gates	36	36
BCD counters	3	3
Monostable timers	6	6
Logic differentiators	6	6

The following table should prove useful to identify the component complement on each of the analog consoles. It should also prove invaluable for producing simulations which can run on any console. Programming in this manner will insure that you will be able to run when any console is available instead of having to wait for a specific console.

	CONSOLE NUMBER			
	A1	A2	B1	B2
ATTENUATORS				
Servo Set Pots				
P00 - P39	X	X	X	X
P40 - P79		X		
P80 - P119	X	X	X	X

Four Quadrant Inverting DCA's				
P40 - P79	X		X	X
Handset Pots				
Q - 2,4,7,9,12,14,17,19,22,24,27,29	X	X	X	X
AMPLIFIERS				
Summers/Integrators				
A - 0,2,5,7,10,12,15,17,20,22,25,27	X	X	X	X
A - 30,35,40,45,50,55,60,65,70,75	X	X	X	X
A - 80,85,90,95,100,105,110,115	X	X	X	X
Inverters				
A - 4,9,14,19,24,29,34,39,44,49	X	X	X	X
A - 54,59,64,69,74,79,84,89,94,99	X	X	X	X
A - 104,109,114,119	X	X	X	X
Multipliers				
A - 3,8,13,18,23,28,33,38,43,48	X	X	X	X
A - 53,58,63,68,73,78,83,88	X	X	X	X
A - 93,98,103,108,113,118	X	X	X	X
A - 92,97,102,107,112,117	X		X	X
T/S Summers/Limiters				
A - 1,11,21,31,41,51,61,71,81,91,101,111	X	X	X	X
Zero Limit Summers				
A - 6,16,26,36,46,56,66,76,86,96,106,116	X	X	X	X
A - 32,37,42,47,52,57,62,67,72,77,82,87	X		X	X
FUNCTION GENERATORS				
DCFG's				
F00 - F07	X		X	X
DFG's (not recommended for use)				
A - 32,37,42,47,52,57		X		
Sine/Cosine Function Generators				
A62, A92		X		

LOGIC

Logic is as shown on patchboard for all patchboards except for: SCOPE, RECORDER, and DIS brown areas; ERS-1, ERS-2, Y1, Y2, Y3, Y4, and D/A transfer control. Also the analog input port to the CRT terminals is available on consoles A1 and B1 only.

	SYSTEM A	SYSTEM B
Analog-to-digital converters (ADC's)	32	48
Digital-to-analog converters (DAC's)	24	24
Control lines on each analog console	16	16
Sense lines on each analog console	8	8
General purpose interrupts per console	8	8
Interface clock	1	1

Note 1: All ADC's are 14 bits plus sign. ADC's do not invert.

Note 2: All DAC's are 14 bits plus sign. DAC's do not invert and all 24 are multiplying DAC's (DAM's).

The following table is also intended to help in producing simulations which can run on any console.

ADC's

Console A1 - Numbers 0-31 as shown on patchboard.
Console A2 - None.
Console B1 - Numbers 0-31 as shown on patchboard.
Console B2 - Numbers 32-47 shown on patchboard as numbers 0-15.

DAC's/DAM's

Console A1 - DAM's #0-23 have inputs/outputs as shown on patchboard.
Console A2 - None
Console B1 - DAM's #0 - 23 have inputs/outputs as shown on patchboard.
Console B2 - None

CENTRAL TRUNKING SYSTEM

The new trunking system for the hybrid facility should offer numerous advantages to the user. The trunking cables have been upgraded to a three wire 22 gauge system. The increased gauge size over the old system should reduce resistive losses and the addition of the third wire should increase system stability through reduced cable inductance and improve problem signal to noise ratio's because of improved shielding. To take full advantage of these benefits, however, the direction of signal flow over your trunks must proceed in the direction we have assigned to each trunk. That is, there are input trunks and output trunks. Trunks are grouped by tens and placed in a cable. All trunks within the cable are treated identically and the cable becomes the fundamental patching unit, each trunk in the cable being patched from the same source to the same destination.

A signal which originates on console number one (from an amplifier output) and is received by console number two should be patched through an output cable on console number one to the trunking station then to an input cable for console number two.

The trunking station consists of two open back cabinets where the trunk cables containing ten signals from each console terminate in highly reliable, rapid disconnect, multi-pin connectors. Patching is accomplished by interconnecting these cables to one another through short "patch" cables which have appropriate mating conectors on each end. When trunk cables from individual consoles are connected together via these "patch" cables, the console's trunks are virtually hard-wired together. The reduced connection points over a conventional patch board system increases the reliability of connection three fold.

The patching of the ten-trunk cables through the terminating connectors makes erroneous cable interconnection highly unlikely. The connectors themselves make it impossible to inadvertenly short a signal by reverse patching (high to low or vice versa) as frequently occurs on conventional patchboads employing a three wire system because of the ease of twisting a three wire patch cord.

The impact of such a system to the user is that he must now group all his signals in blocks of ten, with all signals originating, for example, on console number one and terminating on console number two being placed in the same group. Likewise signals originating on console number two and terminating on console number one should be grouped and placed in a different block of ten trunks. In a similar fashion, signals which originate on console number one and go to two different consoles, for example, console number two and the SEL 810B should be grouped and placed in a different block of ten. The latter block will require splitting the output cable from console number one so that it may go to two input cables, namely one to console number two and one to the SEL 810B. This can be accomplished by using one of the six tie points available on the central trunking station.

Since most problems have a surplus of trunks, wasting trunks by having unfilled blocks of ten should be no problem. Larger simulations, however, may require more careful planning. For those situations where it is impossible to make all interconnections without splitting a group of ten, a conventional AMP patch panel with 12 ten-trunk cables going to the central trunking station is available for cross patching between cables.

A list showing the available trunks at the central trunking station and their arrangement in cables is provided at the end of this section. Notice that the list shows all inputs and outputs available for trunking in the facility. Note also that there is no longer any hard-wired trunks between consoles. Also note that the consoles are numbered one through four instead of by their usual designations A1, A2, B1, and B2. This is because most problems are no longer tied to a given console but can run on either system. Hence, when you sign up for a console, the available consoles will define which is number one, which is number two, etc.

A user conveys his patching configuration to the hybrid facility technicians (who will do the patching) via a trunking setup sheet or block diagram. The user should fill out one or the other for his problem. The setup sheet or block diagram should specify the user's patch panel number(s). This information, together with the computer schedule board, which assigns the patch panel to a particular console, allows for the proper patching of facility trunks by the technicians.

J.O. _____

DATE _____

AVAILABLE TRUNKS - CENTRAL TRUNKING STATION

CONSOLES #1-4 (A1, A2, B1, B2)			TR48 #1+2			TR48 #3		
INPUTS	OUTPUTS		INPUTS	OUTPUTS		INPUTS	OUTPUTS	
70-79	0-9		15-24	0-9		30-39	0-9	
80-89	10-19		25-33	10-14		40-46	10-19	
90-99	20-29						20-29	
100-109	30-39							
110-119	40-49							
AUX 0-9	50-59							
	60-69							
	AUX 10-17							
	REC #1							
	REC #2							

SEL 810B			PRESTON AMPS			MAG TAPE	
INPUTS	OUTPUTS	Bi-DIRECTIONAL	INPUTS	OUTPUTS		INPUTS	OUTPUTS
0-9	0-9	UTILITY 0-9	0-9	0-9		0-9	0-9
10-19	10-19	UTILITY 10-15	10-19	10-19		10-13	10-13
20-29	20-29		20-29	20-29			
30-39	30-39		30-39	30-39			
40-49	40-49		40-49	40-49			
50-59	50-53		50-59	50-59			
60-63	REC #1		60-63	60-63			
70-79	REC #2						
80-83	REC #3						

J.C. _____

DATE _____

AVAILABLE TRUNKS - CENTRAL TRUNKING STATION

	AD-YU		BRUSH RECORDERS (EACH CABLE 0-7)		AMP CROSSPATCH PANEL (EACH CABLE 0-9)	CABLES TO HOT TURBINE FACILITY (ERB)	CABLES TO PSL 1 & 2	CABLES TO PSL 3 & 4	CABLES TO 10 x 10
	INPUTS		CONSOLE #1		CABLE #1	0-9	0-9	0-9	0-9
	0-9		#1		" #2	10-19	10-19	10-19	10-19
			#2		" #3	20-29	20-29	20-29	20-29
	OUTPUTS				" #4	30-39	30-39	30-39	30-39
	0-9		CONSOLE #2		" #5	40-49	40-47	40-47	40-47
			#1		" #6	50-59			
			#2		" #7	60-69			
					" #8	70-79			
			CONSOLE #3		" #9	80-89			
			#1		" #10	90-99			
			#2		" #11				
					" #12				
			CONSOLE #4						
			#1						
			#2						
			SEL BUS B						
			#1						
			#2						
			STATION #11						
			#1						

NASA-C-801B (10-24-51)

How To Use The Trunking System

To help eliminate confusion which might occur when using the system for the first time, we offer the following organizational procedure for developing your trunking system. First, list all signals which need to be trunked. Second, list next to each signal its source and all its ultimate destinations. If all signals which will reach a destination must go through a common point for pre-processing (such as an attenuation amplifier before going to the tape recorder) only the initial destination need by listed (ie. the attenuator amplifiers), as the remaining destinations will automatically follow. Third, group the signals which follow a common source-to-destination path. At this point drawing a diagram similar to figure 1 showing the number of trunks in each group will be helpful.

You are now ready to assign the signals to the trunks. Use a new cable of ten trunks for each new group. A good method for keeping track of which signals are on which trunks is to construct a table such as the one provided at the end of this section. Making assignments on such a table will further ease cable layout, as the table clearly shows how signals can be grouped into cables and which cables have unused trunks available for further expansion. Furthermore, this table, coupled with your diagram, will provide good documentation of your system for tracing signals.

If you run out of cables at either a source or destination, you will have to split a cable which has some unused trunks. This was the case in figure 1 where the AMP crosspatch panel was used for splitting a cable from PSL. If you avoid assigning signals to such split cables until after all groups of ten have been assigned, it will become obvious which signals you want to put on the split cable. Becase of potential conflicts with other users and the additional patching required, we discourage use of the amp crosspatch panel unless absolutely necessary. Rather we encourage use of tie points to route a source to two different destinations. This may result in sending signals to places where they aren't needed, but this is of no concern as long as there are sufficient trunks at the destination to handle the unused signals. In the example of figure 1, the crosspatch panel was required because both the PSL 1 & 2 cables and the SEL Utility cables were fully used. In general, this should be a rare situation. We must also mention that no major catastrophy will occur if a trunk within a cable is used backwards, that is using an output cable for an input. The practice should be avoided, however, as differential amplifers may pick up millivolt offsets.

Lastly, if you want the facility technicians to patch your problem for you when you sign up for a console, you will have to translate the cable assignments to a trunk setup sheet or block diagram as shown in figure 1.

J.O. _____

DATE _____

EXAMPLE OF TRUNK ASSIGNMENTS FOR MULTI-CONSOLE PROBLEM ENG. SHEET 1 OF 2

	SOURCE →				DESTINATION →			
	SIGNAL	PSL	SEL			PSL	SEL	MAG TAPE	
	START TRAN	0					0	0	
	PLA	1					1	1	
	NL	2					2	2	
	NH	3					3	3	
	CIVV	4					4	4	
	PB	5					5	5	
	FTIT	6					6	6	
	RCUV	7					7	7	
	TT2	8					8	8	
	TT25C	9					9	9	
	PT25C	10					10	10	
	PT25H	11					11	11	
	DP25C	12					12	12	
	DP25H	13					13	13	
	PT2	14					14	14	
	WFFB	15					15		
	PT6C	16					16		
	AJ	17					17		
	PT6H	18					18		
	PB	19					19		
	ABLC	20					20		
	FAEN	21					21		
	WFTOT	22					22		
	PT5AI	23					23		
	RUN MODE		U-10			24			
	TRACK		U-11			25			
	FREEZE		U-12			26			
	DP/PTRIM		U-13			27			
	NL TRIM		U-14			28			
	FTIT TRIM		U-15			29			
	PB LOW TRIM		U-0			30			

NASA-C-8018 (10-24-51)

J.O. _____

DATE _____

EXAMPLE OF TRUNK ASSIGNMENTS FOR MULTI-CONSOLE PROBLEM ENG. SHEET 2 OF 2

	SOURCE →				DESTINATION →			
SIGNAL	PSL	SEL			PSL	SEL	MAG TAPE	
WF FLAG		U-1			31			
BLEED FLAG		U-2			32			
AJ FLAG		U-3			33			
RCVV FLAG		U-4			34			
CIVV FLAG		U-5			35			
WF TRIM LIMIT FLAG		U-6			36			
AJ TRIM LIMIT FLAG		U-7			37			
RELAY CLOSE		U-8			38			
SEL BIOD SCRAM		U-9			39			
SEL BIOD AC		0-0			40			
AJ BLCOM		0-1			41			
AJCOM		0-2			42			
CIVV CM		0-3			43			
WFCOM		0-4			44			
RCVV CM		0-5			45			
DP2SE		0-6			46			
DP2ST		0-7			47			

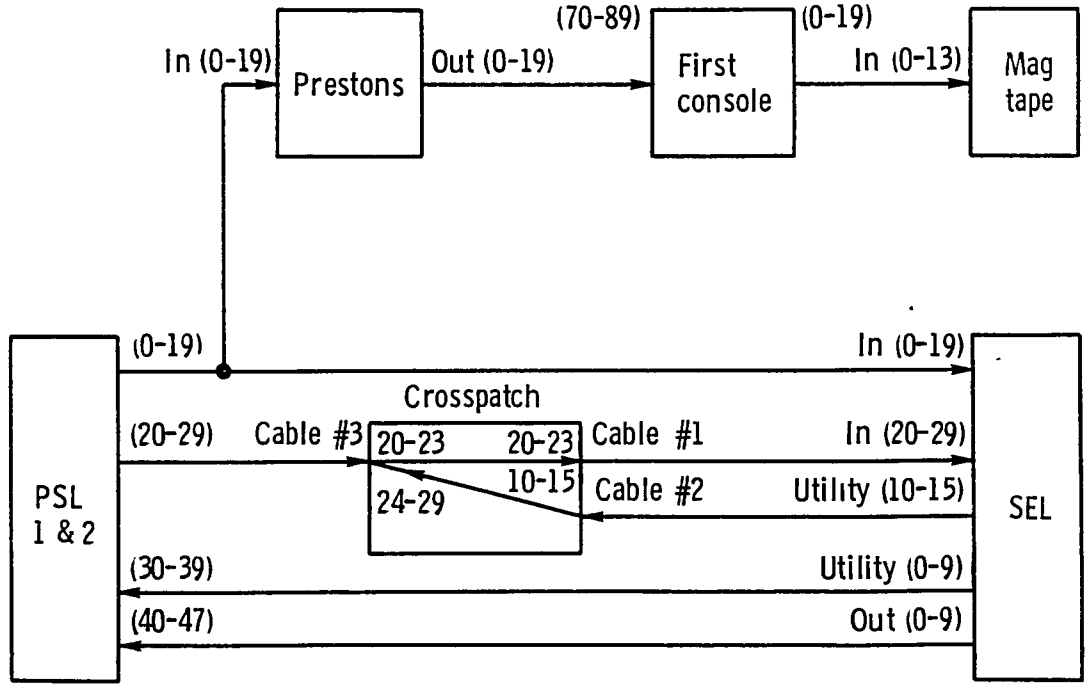

Figure 1. - Patching block diagram for a typical job requiring splitting of a cable.

Brush Recorders

With the new trunking system you are no longer restricted to two Brush recorders per console. Normally, however, you will use the two recorders adjacent to a console as the Brush recorders for your problem before expanding to more recorders. Hence, these recorders will normally be patched as if they were hard-wired. When you run your problem on different consoles, you will automatically shift to the Brush recorder adjacent to that console without having to patch anything on the central trunking system (assuming you used the recorder input holes on the console's patch board). If you use more than two recorders or if you want the four recorders associated with a two console problem to be patched as three on the first console and one on the second, you will have to indicate same on the trunk setup sheet. The normal Brush recorder patching is permanently shown on these sheets. To request the removal of a normally patched cable, draw a circle around the dot and put an X through the circle on the trunk setup sheet.

If you have a one console problem but need three recorders you will have to "steal" one from an available console. Hence, you should fill out the setup sheet as if you were using a recorder from console number two. Likewise you should "steal" the additional recorder from console number three if you have a two console problem, etc. Which recorder you actually get each day you run will depend on which recorder is available at the time. The auxiliary 10-17 output cable from an analog console is ideally suited for use in trunking to the third Brush recorder.

The dual disk system is a two platter system with both hardware and software protect features. Each platter has a hardware protect switch which, when on, prevents any output (writing) to that disk. One platter (platter number 1) is a removable disk cartridge, which contains user-derived files, and is normally not hardware protected (switch off). The other platter (platter number 2) is a fixed disk platter, which contains system files, and is normally hardware protected (switch on). Software protection is provided by MONITOR commands to be discussed later.

Associated with the disk is what is known as the Moving Head Disk Operating System (MHDOS). This software system consists of a system loader, MONITOR, MONITOR input/output (I/O) routines and system programs.

The disk MONITOR is the executive system routine that enables the user to direct control of the digital processor through the CRT terminal keyboard or the card reader. The MHDOS also includes the Control Option Processor (.COP), a non-resident control command processing program capable of interpreting control command records to set processor and I/O options, load programs, and execute them as required.

The system programs consist of language processors, run-time library (RTL), debugging aids, diagnostics, and various utility routines. These programs are available in two types of format, Core Image (CI) and Relocatable Object (OB). CI programs can be loaded directly into memory by the MONITOR and occupy fixed memory locations. The Core Image Generator (CIG) may be used to generate a CI version of the relocatable programs which MONITOR can load directly.

FILE TYPES

The system programs and user-derived programs exist on disk as named files. In addition to Core Image (CI) and Relocatable Object (OB) files, Source (SO) and Data (DA) files may also be output to disk by the user. The various file types are referred to in the software by number. The file type numbers are given in the following table:

FILE TYPE	FILE TYPE NUMBER
Source (SO)	0
Relocatable Object (OB)	1
Core Image (CI)	2
Data (DA)	3

The file type numbers, which must be entered by the user for some commands, cannot be interchanged. That is, a CI file is always type 2.

DEVICE NUMBERS

It is sometimes necessary for a user to specify a device number (or a logical unit number in the case of disk files) when entering a command. The following table lists the peripheral devices and the corresponding logical unit number. While all numbers are octal numbers, they are not entered as octal (i.e., no apostrophy preceding the number) when using the operating system programs.

DEVICE	LOGICAL UNIT NUMBER
CRT screen	1
CRT keyboard	2
Paper tape reader	4
Paper tape punch	5
Card reader	6
Line printer	20
Disk (logical units)	21-24

While the disk logical unit numbers can sometimes be used interchangeably, it is recommended that users adhere to a specific convention. In the following table, the recommended disk logical unit number for each type of file is listed.

FILE TYPE	LOGICAL UNIT NUMBER
Core Image (CI)	21
Relocatable Object (OB)	22
Source (SO)	23
Data (DA)	24

1. Press the 681 (680) analog "POWER ON" button. The analog console will go to Pot Coefficient mode (PC).
2. Press the "ENG" button to insure that the patch boards are properly engaged.
3. Check the "REF", "MODE", and "DIGITAL COMPUT I/O" slave switches located behind logic readout panel. Select the proper combination.
4. Press the digital mode control "R". In this mode all logic elements (such as "AND" gates and comparators) operate normally.
5. Select the analog time scale by pressing "SEC" and either "N" or "F" ("MS" is normally used for analog Rep-op problems). See reference 1 for further explanation.
6. Turn on the PACER digital unit. Be sure that "EXECUTE/RUN" and "RESET I/O" and "COMP" have put the digital in the "REST" mode. If the computer is not in "REST", be sure that "EXECUTE/RUN" is in the "DOWN" position and that "RESET I/O" and "COMP" have been momentarily depressed.
7. Turn on the CRT (switch is under keyboard).
8. Turn on the disk unit. Place it in the "READY" mode by depressing the "RUN" button. About two minutes is needed to get the "READY" light.

SYSTEM SHUTDOWN

1. Place the disk in the "SAFE" mode by depressing the "STOP" button. Wait about 20 seconds for the disk "SAFE" light to come on.
2. Be sure the digital is in the "REST" mode. Turn off the disk and digital.
3. Turn off the CRT (and Hard Copier, if on).
4. Turn off the analog consoles.

MONITOR ENTRY

1. Place the disk in the "READY" mode by depressing the disk "RUN" button.
2. Release the "EXECUTE/RUN" switch (down position).
3. Momentarily depress the "RESET I/O" and "COMP" switches.
4. Enter '77777 in the P register
5. Place "EXECUTE/RUN" in the up position.

NOTE: If MONITOR appears to be "bombed", double check that the disk is "READY" before referring to page 2-1 in reference 2 for using the Bootstrap Loader to reload MONITOR.

MONITOR COMMANDS

In the following table, the more common MONITOR commands, with examples, are listed and briefly described. Additional commands and more detail may be found in chapter 2 of reference 2.

In the table below, LU refers to the disk logical unit number; ADR to the starting octal address of the program; DEV to the peripheral device number; X,A,Q,K (in #R command) to the four hardware registers which have to be set prior to executing some system programs; and T to the file type. All names may consist of up to six alphanumeric characters.

MONITOR COMMAND (EXAMPLE)	FUNCTION
#L,NAME,LU (#L,JONES,21)	Locate and load CI file (named JONES)
#G,ADR (#G,1000)	Execute program in core (starting at '1000)
#X,DEV (#X,2)	Transfer control to .COP (at keyboard)
#R,X,A,Q,K (#R,,2,1)	Set hardware registers (X=0,A=2,Q=1,K=0)
#S,NAME,LU (#S,JONES,21)	Secure CI file (named JONES) - software protect
#U,NAME,LU (#U,JONES,21)	Unsecure CI file (named JONES) - unprotect
#P,NAME,LU (#P,SMITH,23)	Position to a source file (named SMITH)
#N,NAME,LU,T (#N,SUB,23,0)	Create a new source file (named SUB)
#D,ADR1,ADR2,LU (#D,0,23141,21)	Dump memory (between 0 and 23141) in CI format
#C,LU (#C,23)	Close the open file (source)
#D,DEV (#I,6)	Change input device (to card reader)

With the above MONITOR commands, the user can load and execute a CI file. However, if he desires to create a new program, considerable effort can be avoided by using the Control Option Processor (.COP) for compiling (or assembling) and forming CI files.

.COP Standard Options

Source input device for Compiler or Assembler - Card Reader
Source listing device for Compiler or Assembler - Line Printer
Object output device for Compiler or Assembler - Disk LU 22
Object input device for Core Image Generator (CIG) - Disk LU 22
Core image output device for CIG - Disk LU 21

COMPILER OPTIONS

Source statement listing
Map listing
Error message listing
No symbolic listing
Write object output on disk
In-line assembly processor active
Card reader driver active
Paper tape input driver active
Format error scanner active
No in-line assembly coding for integers and scaled fractions
No in-line assembly coding for floating point processor

ASSEMBLER OPTIONS

No pause between passes
Scratch file on disk LU 24

CORE IMAGE GENERATOR (CIG) OPTIONS

Zone Zero base starts at '00000
Top of common is '60000
Program execution address is '1000
Memory map listing included (set SSW A to suppress)

Exceptions to the above .COP options can be made by the user for a specific job by issuing .COP control commands. In the following table the more common .COP control commands, with examples, are listed and briefly described. Additional commands and more detail may be found in chapter 3 of reference 2.

.COP COMMAND (EXAMPLE)	FUNCTION
$JOB ($JOB)	Initialize .COP for batch job
$IN,DEV ($IN,4)	Specify input device (as Paper Tape Reader)
$IN,LU,NAME ($IN,23,SMITH)	Specify input device (as disk) with source file name (SMITH) for use with Compiler or Assembler
$OUT,LU,NAME ($OUT,21,SUB)	CI file (named SUB) is created, identified, and positioned
$FOR,"OPTION STRING" ($FOR,NOL,NOM,NOO)	Compile (with no Source list, no Map, and no Object, i.e., Error list only)
$FOR NAME,"OPTION STRING" ($FOR SUB,I)	Compile (with in-line assembly coding for Integers and Scaled Fractions; file name will be SUB)
$FILE,LU,NAME ($FILE,22,LINKN)	Specify file other than RTL for CIG loading; (specify linkage routines)
$LOAD ($LOAD)	Create a Core Image file
$RENAME,LU,NAM1,NAM2 ($RENAME,21,CRMG,MINE)	Change name (of disk file CRMG to MINE) (cannot be used for secured files; use #U MONITOR command first)
$MON ($MON)	Return control to MONITOR.

No more than five $FILE commands may be given for a specific job. If more than five files, in addition to the main program and the RTL, are required, .COP cannot be used. See section 2.9 of reference 2 for using the CIG directly.

While the $OUT command (and $FOR for Object files) can be used to name Object and Core Image files. It is normally recommended that names not be specified initially. If no name is specified, the default names; OBJTAA through OBJTZZ for Object files, and CRMGAA through CRMGZZ for Core Image files, will be assigned by .COP. If it is then determined that the file is to be retained, the file can be renamed with the $RENAME command. The primary reason for taking this approach is to simplify the disk housekeeping chores to be discussed later.

Following a $LOAD (.COP command) the Core Image Generator (CIG) is loaded by .COP and control passes to the CIG. If the CIG detects an error, control passes to the user at the keyboard. This is indicated by the printing of an asterisk (*) on the CRT terminal. The user must, before exiting from the CIG, close the output file with a #M command.

MONITOR AND .COP SYSTEM RESPONSES

RESPONSE	MEANING
M_	MONITOR entered and waiting.
LINKN OB P2	Positioning successful for Object file, LINKN, on platter 2.
YOURS SO P1	Positioning successful for Source file, YOURS, on platter 1.
.COP CI P1	Positioning successful for Core Image file, .COP, on platter 1.
LD	Successful load.
C_	.COP entered.
!	.COP waiting.
FR	Free area of disk found.
?	File not found, duplicate file name, or command error.
CE	Core Image error (wrong type file following #L command).
DE	Disk hardware error, try again.
NR	Card Reader not ready.
KB	Keyboard control.
PV	Protect violation.
UN	Unit error (not 21,22,23, or 24)
OP	Last file on platter still open.
CL	The last file left open was just closed.

See chapter 2 of reference 2 for additional error and recovery procedures for MONITOR and the CIG, and chapter 3 for .COP messages.

INTERACTIVE UTILITY PROGRAM (MIU)

The Interactive Utility (MIU) program is used primarily for disk housekeeping chores. If some of these chores are not taken care of, the disk space would rapidly be used up. MIU can be used for many functions, However, the average user need only concern himself with a few of the functions; listing files on disk, deleting files that are no longer of value, and possibly providing a user identification for files.

The loading and executing of MIU is accomplished with MONITOR commands as follows:

```
#L,MIU,21          Load MIU.
#R,,2,1            Set required registers.
#G,1000            Execute MIU.
```

After MIU is entered, the user responds to messages at the CRT terminal. The user response to the first message informs the MIU control program that the disk is on the Direct Memory Access channel (DMAC). The initial message and response is:

```
DMAC?              MIU request.
D                  User response.
```

MIU then asks the user what task is to be performed and on what logical unit the desired function subroutine can be found.

If the user wanted to list all files on platter 1 at the CRT terminal, the messages and user responses would be:

```
TASK DESIRED,UNIT        MIU request.
LIST,21                  User response for list.
FROM,TO                  MIU request.
D,21,1                   User response (from disk to CRT).
WHICH PLATTER            MIU request.
1                        User response for platter 1.
```

The LIST subroutine of MIU would then list all files by name, giving file type number, track number, and sector number. If the user just wanted to list those files following and including a particular file, named HIS for example, the message and response would be:

```
TASK DESIRED,UNIT
LIST,HIS,21
```

Similarly if it was desired to list only those files between and including HIS and HERS, the message and response would be:

```
TASK DESIRED,UNIT
LIST,HIS,HERS,21
```

If the user now wanted to delete a file, named HIS, from platter 1, the messages and responses would be:

```
TASK DESIRED,UNIT      MIU request
DEL,21                 User response for delete
FROM                   MIU request
D                      User response for disk
WHICH PLATTER          MIU request
1                      User response for platter 1.
NAMES TO BE DELETED    MIU request
HIS                    User response to delete file HIS.
```

One important point to note here is that any time a delete command is given, all files given default names; OBJTAA through

OBJTZZ and CRMGAA through CRMGZZ, by .COP, are also deleted. If more than one named file is to be deleted, up to eight may be deleted at once by responding with a string of names. For example, the message and response to delete four files might be:

```
NAMES TO BE DELETED
HIS,HERS,MINE,YOURS
```

User Identification of Files

Two ways to protect disk files have been discussed. Hardward protection (via the protect switches) will not allow anything to be written or changed on disk. Software protection is provided by the #S (secure) MONITOR command and prevents inadvertent writing on the protected file. However, it does not prevent deletion of the file with the MIU delete function. A safeguard against this possibility is provided with the use of a user identification for a file.

To provide the user identification, MYFILE, for a file, named SUB, on platter 1, the messages and responses would be:

TASK DESIRED,UNIT	MIU request
USERID,21	User response for ID.
WHICH PLATTER	MIU request
1	User response for platter 1
FILES TO BE GIVEN USERID	MIU request
SUB	User response for file SUB.
USER ID FOR FILES	MIU request
MYFILE	User response with ID for file SUB.

Once a file has a user ID, it can only be deleted by responding to a request for the user ID by MIU when using the delete function. It should also be noted that a user ID can only be changed by deleting the file and placing it back on the disk.

It is up to the user whether or not to provide user ID for his files. However, it is generally not considered necessary.

Return to MONITOR

At the completion of each MIU task, MIU requests a new task. To return to MONITOR, the message and response is:

```
TASK DESIRED,UNIT          MIU request
MO,21                      User response to return to MONITOR.
```

For additional information on MIU, see chapter 11 of reference 3.

OEDIPUS DEBUG PROGRAM (OED)

The OEDIPUS DEBUG program is a utility routine operated on-line in a conversational mode. Commands from the CRT terminal provide for storage readout, correction, search, save and dump, trap setting, and program loading.

The loading and execution of OEDIPUS DEBUG is accomplished with MONITOR commands as follows:

```
#L,OED,21          Load OEDIPUS DEBUG.
#G,70100           Execute OEDIPUS DEBUG
```

OEDIPUS types "OK" and waits for a terminal command from the user. Commonly used commands, where N is an octal argument, are:

N:	Open a cell and display it.
N;	Open and display relative-addressed cell
.	Open next cell on next line.
,	Open next cell on same line.
-	Open previous cell
NG	Go to address N.
G	Execute from last trap encountered
NA	Set accumulator to N
NQ	Set Q register to N.
NX	Set Index register to N.
NK	Set K register to N.
SPACE BAR	Display A, Q, X, and K registers.
NL	Set lower limit of block to N.
NU	Set upper limit of block to N.
NB	Set base address.
NZ	Set zone zero location for trap pointer.
NT	Set trap at N.
NW	Wipe out trap at N.
W	Wipe out all traps.
ND	Dump with N columns
RUB OUT KEY	Delete current digit string.

For additional information on OEDIPUS DEBUG, see chapter 6 of reference 3.

HYBRID DEBUG PROGRAM (HYDBG)

HYBRID DEBUG is a utility routine which serves as a debugging aid for the Hybrid user. It is similar in many respects to OEDIPUS DEBUG. The following functions may be performed: storage readout, storage dump, storage correction, storage search, breakpoint set, load DAC, list ADC or sense lines, set or reset control lines, read or set pots and DCA's, and program timing.

The loading and execution of HYBRID DEBUG is accomplished with MONITOR commands as follows:

```
#L,HYDBG,21          Load HYBRID DEBUG.
#G,61000             Execute HYBRID DEBUG.
```

HYBRID DEBUG outputs a carriage return and line feed and waits for a terminal command from the user. Commonly used digital commands, where N is an octal argument, are:

```
   B          Clear breakpoint
   D          Dump memory between $B and $E (8 per line).
   LF         Open next cell on new line
   N:         Open and display N.
  N,B         Set breakpoint at N.
  N,G         Transfer control to N.
RUB OUT       Delete characters.
  $B:         Set lower limit of Block.
  $E:         Set upper limit of block.
"Up Arrow"    Display previous cell.
   /          Display effective address of opened cell.
```

Commonly used Hybrid commands are:

```
@ADXX,L            List value of ADC channel XX.
@CKKK=.XXXX        Set pot KKK to the value .XXXX
@EKKK=-.XXXX       Set DCA KKK to the value -.XXXX
@DAXX=.LLLL        Output to DAC channel XX the value .LLLL
@1,K               Select console 1.
@XXXXX,YYYYY,E     Time program execution from XXXXX to YYYYY.
```

For additional information on HYBRID DEBUG, see chapter 7 of reference 3.

OTHER EAI PROGRAMS

Basic Text Editor (BTE) - Discussed in reference 3, chapters 1-5.

Hytran Operations Interpreter (HOI) - Discussed in reference 4, chapters 11-15.

Core Image Generator (CIG) - Discussed in reference 2, chapter 2.

PROGRAM NAME	LOAD AND EXECUTE COMMANDS	RETURN TO MONITOR COMMANDS
.COP	#X,2	$MON
MIU	#L,MIU,21 #R,,2,1 #G,1000	MO,21
HOI	#L,HOI,21 #R,,,70000 #G,1000	R;
BASIC TEXT EDITOR	#L,BTE,21 #R,,,170000 #G,1000	M
If editing from/to disk do:		
	#L,BTE,21 #R,,,170000 #P,OLDNAME,22 #N,NEWNAME,23 #G,1000	Note that these unit numbers must differ
OEDIPUS	#L,OED,21 #G,70100	77777,G
HYBRID DEBUG	#L,HYDBG,21 #G,61000	77777,G
SETUP	#L,SETUP,21 #G,1000	End of program or PACER control panel
GENERAL USER PROGRAM	#L,NAME,21 #G,1000	"Call MONOUT" or PACER control panel

Once a user's program has been loaded, reentry to his program can be accomplished at the digital console without returning to MONITOR. Simply enter '1000 in the P register and place the "Execute/Run" switch in the up position. It is recommended that when the disk is no longer needed, that it be placed in the "SAFE" mode.

LEWIS FACILITY SOFTWARE

Bivariate-Function Routine (MAP, MAPL)

Statistics

Name: File- MAPS

Subroutines- MAP, MAPL, MOOR, DATAIN

Common - MAPDAT,MAPVAL

Type: Fortran callable subroutines

Location: PACER A and B Moving Head Disks

Length: (MAP,MAPL) - '322

MOOR - '164

DATAIN - '1655

MAPDAT - Variable

MAPVAL - '3

Programmer: William M. Bruton

Source: (MAP,MAPL) - Assembly language cards

MOOR, DATAIN - Fortran cards

Date: August 1978

Subroutines Required: (MAP,MAPL) - MOOR

MOOR- LINKN and Fortran RTL libraries

DATAIN - Fortran RTL libraries.

Loading Sequence: MAPS, LINKN, the Fortran RTL Libraries.

Description

A new fortran callable function subprogram has been written to replace "MAP2" for generating functions of two variables. This new program, called simply "MAP", is more general than "MAP2" in that it does not make use of fixed data array sizes. The user specifies the size of all data arrays or tables.

"MAP" may be used for any function of two variables, but was written specifically to handle functions that cannot be defined over a rectangular array. That is, each curve, Y, does not have to extend over the entire range of points, X. "MAP" performs

radial interpolation of the map data and updates MAP interpolation search indices prior to returning to the calling program. Upon reentry for each map, the search begins in the X and Y intervals previously used for that particular map.

Use

Some basic requirements of "MAP" follow. (1) Each curve of a given map must have an equal number of breakpoints. The assumption is then made that corresponding points on the curves are functionally related. (2) For each map, the curves, (Y values) and for each curve, the points (X values) must be entered in increasing order. (3) Consecutive scaled values of X, Y, and the output, Z, must differ in magnitude by less than 1.0. This applies to consecutive values of X and Z on the same curve as well as X and Z values in corresponding positions on consecutive curves.

If more than one function (i.e., Z1, Z2, ..., Zn) with common inputs, X and Y, are required, a second entry point to "MAP", called "MAPL", may be used. This applies only if identical X and Y values are used. Considerable processor time and core requirements can be saved by using "MAPL". If "MAPL" is used, it must be called before calling "MAP" for a different function. "MAPL" looks up and calculates function outputs only. No searching is done. It depends on the preceding entry to "MAP" to find the correct table entries. The number of common functions is limited only by the amount of computer memory available.

The computation time for one "MAP" function is 280 microseconds if the inputs remain in the same data intervals. If an input moves to a new interval, 20 to 30 microseconds of additional time is required per interval change. The computation time for "MAPL" is 80 microseconds.

For each map (i.e., set of X and Y input values), two arrays must be declared. The first array is an integer table of length 5. If two or more maps are involved, then an equal number of integer arrays must be declared. These arrays must be dimensioned consecutively (i.e., no other dimensions in between these arrays).

The second array for each map is a scaled fraction data table. The length of this table depends on the number of curves (NCV), number of points per curve (NPT), and the number of functions with common inputs (NFCT). The table size for each map can be determined by using the following formula:

TABLE SIZE = NCV * (NFCT * NPT + NPT + 1)

If two or more maps are involved, then, as for the integer arrays, an equal number of scaled fraction arrays would have to

be declared consecutively with no other scaled fraction declarations in between the arrays.

In addition to any problem variables that are declared scaled fractions, MAP and MAPL, if used, must also be scaled fractions.

The call to the subprogram "MAP" contains four arguments which are in order; the integer array, the scaled fraction array, the X input variable, and the Y input variable. For example:

```
ZB1 = MAP(NB,FB,XB,YB)
```

If a second function with common inputs is included, only one argument is contained in the call, namely the scaled fraction array. For example:

```
ZB2 = MAPL(FB)
```

To be used in conjunction with "MAP" and "MAPL" are two subroutines; "DATAIN", which is called to read map data from cards, and "MOOR", which is called to determine what map, if any, has gone out of range.

Data Input Subroutine (DATAIN)

The call to the subroutine, "DATAIN", contains two arguments, the integer array of the first map and the scaled fraction array of the first map. For example:

```
CALL DATAIN(NA,FA)
```

"DATAIN" reads cards which, in addition to the unscaled map data, includes the size of the map array, the scale factors of the data, and the formats in which the scale factors and data are to be read. The subroutine then scales the data and fills the integer and scaled fraction arrays for each map. It continues to read maps until a blank card is read which initiates a return to the calling program. The subroutine can accommodate up to 25 curves per map, 25 points per curve, and 4 functions with common inputs. An example of map data follows. A description of what is on each card follows the example.

```
MAP NUMBER ONE
```

```
1  3  5  1
(3F8.1)  (5F8.1)  (3F8.1)  (5F8.2)
50.0 20000.0      0.1
4000.0  8000.0 12000.0                          Y  VALUES
0.0     20.0     35.0     40.0     45.0         X  VALUES - CURVE 1
0.02     0.02     0.02     0.01     0.00        Z1 VALUES - CURVE 1
0.0      5.0     15.0     27.5     40.0         X  VALUES - CURVE 2
0.04     0.04     0.04     0.02     0.00        Z1 VALUES - CURVE 2
0.0      5.0     10.0     15.0     20.0         X  VALUES - CURVE 3
0.06     0.06     0.04     0.02     0.00        Z1 VALUES - CURVE 3
```

MAP NUMBER TWO

```
2  3  5  3
(5F8.1)  (5F8.1)  (3F8.1)  (5F8.1)  (5F8.2)  (5F8.3)
1.0      1.0      1.0      1.0      1.0
0.2      0.4      0.6                         Y  VALUES
0.0      0.2      0.3      0.4      0.5        X  VALUES - CURVE 1
0.3      0.3      0.2      0.1      0.0        Z1 VALUES - CURVE 1
0.15     0.15     0.10     0.05     0.00       Z2 VALUES - CURVE 1
0.225    0.225    0.150    0.075    0.000      Z3 VALUES - CURVE 1
0.0      0.4      0.5      0.6      0.7        X  VALUES - CURVE 2
0.6      0.6      0.4      0.2      0.0        Z1 VALUES - CURVE 2
0.30     0.30     0.20     0.10     0.00       Z2 VALUES - CURVE 2
0.450    0.450    0.300    0.150    0.000      Z3 VALUES - CURVE 2
0.0      0.6      0.7      0.8      0.9        X  VALUES - CURVE 3
0.9      0.9      0.6      0.3      0.0        Z1 VALUES - CURVE 3
0.45     0.45     0.30     0.15     0.00       Z2 VALUES - CURVE 3
0.675    0.675    0.450    0.225    0.000      Z3 VALUES - CURVE 3
```

MAP NUMBER THREE

```
3  3  5  1
(3F8.1)  (5F8.2)  (3F8.1)  (5F8.1)
1.0      1.0      1.0
0.2      0.4      0.6                         Y  VALUES
0.00     0.30     0.55     0.70     0.90       X  VALUES - CURVE 1
0.2      0.2      0.2      0.2      0.0        Z1 VALUES - CURVE 1
0.00     0.20     0.30     0.55     0.80       X  VALUES - CURVE 2
0.4      0.4      0.4      0.2      0.0        Z1 VALUES - CURVE 2
0.00     0.10     0.20     0.30     0.40       X  VALUES - CURVE 3
0.6      0.6      0.4      0.2      0.0        Z1 VALUES - CURVE 3
```

Note: A blank card must follow the last data card of the last map.

The first card for each map contains the map number, the number of curves, the number of points per curve, and the number of common functions in 4I3 format.

The second card for each map contains the formats for the

scale factors, the X input values, the Y input values, and the Z output values. The format of the card is 16A2 for one function, 20A2 for two functions, 24A2 for three functions, and 28A2 for four functions.

The third card for each map contains the X, Y, and Z scale factors in the format as specified on card two.

The remaining cards for each map contain the Y values, X values, and Z values in formats as specified on card two.

One important consideration, when preparing data, should be noted. Data blocks should not be merged on the same card. Referring to the example, if the X values of map number one had been read in a 4F8.1 format (instead of 5F8.1), the first four values would appear on one card and the fifth value on a second card. The Z1 values would then start on a third card.

If functions arise where the data is such that for each curve the points may be defined by identically the same X values, these values need only be included once, immediately following the Y values. To identify this case for "DATAIN", a non-zero integer in I3 format should be added to the first card of the map data. The first card for that map would then be in 5I3 format.

This will save some cards, but it will not reduce the scaled fraction data table size. That is, "DATAIN" will fill the data table with one X value for each Z1 value. This is necessary because of the manner in which "MAP" manipulates the data.

Map Out of Range Subroutine (MOOR)

The subroutine, "MOOR", is called both by "MAP" and the user's program. It is used to inform the user when a map input has gone out of range and which map it is.

Some logic patching on the analog board is required for "MOOR" to function. The specific patching is shown in figure 2.

If a map variable has exceeded the data range, it is called to the attention of the user by the BCD counter lamps flashing on the analog logic indicator panel. To determine which map has gone out of range, the user pulses a logic pushbutton (PB5) "OFF" at which time the map number and values of the input variables are displayed on the CRT screen. Note that once the CRT screen is filled, the terminal will either overwrite or, if the "BUSY" switch is on, lock up the digital computer. To avoid this problem when running a simulation, it is recommended that the "BUSY" switch be "OFF" and the "BREAK" button be pushed which allows the terminal to "AUTOPAGE" or clear the screen when it is full.

Two of the logic components illustrated must be specifically used, control line 9 (CL9) and sense line 6 (SL6). SLx may be

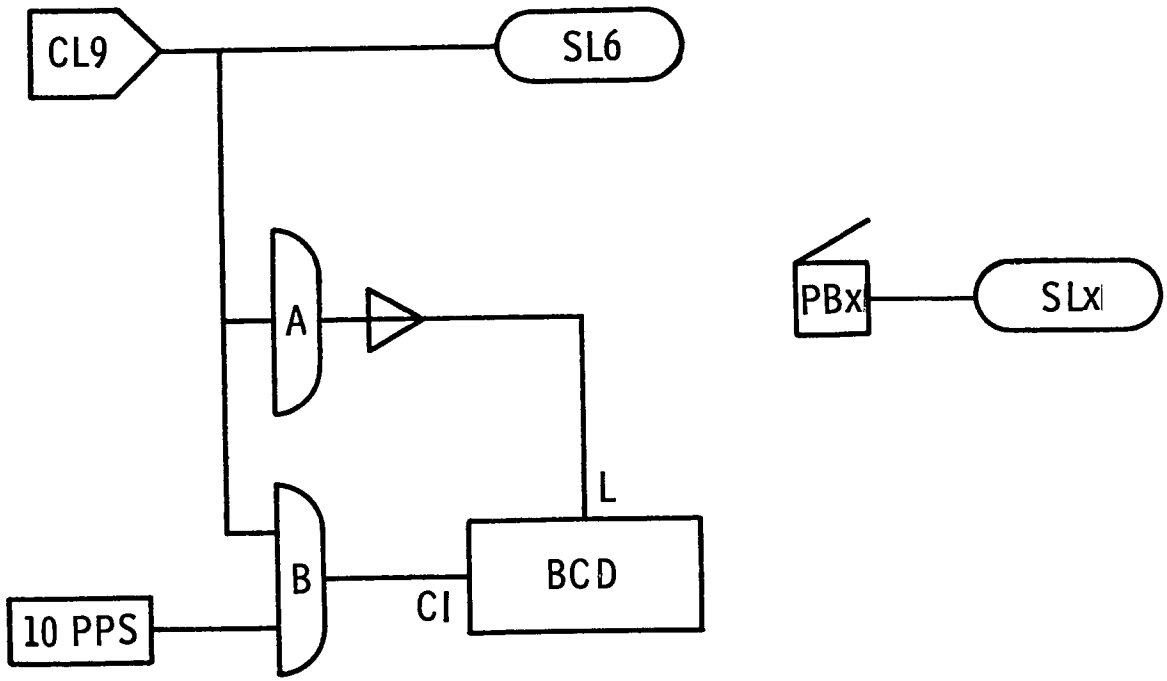

Figure 2. - Logic patching for subroutine MOOR.

any other sense line, but it must correspond to the coding in the user's program which will be discussed later. The "AND" gates, A and B, may be any "AND" gates, the BCD counter may be any counter, and PBx may be any logic pushbutton on the console. The 10 PPS (10 pulses per second) terminal on the patch panel is recommended so that the lamp flashing of the BCD counter is more prominent.

Control line 9 and sense line 6 are used in "MOOR". When "MOOR" is called by "MAP", sense line 6 is tested. If no map out of range message is pending, sense line 6 is "LOW" and control line 9 is set which initiates the BCD counter and sets sense line 6 "HIGH." The map number and input variables are stored. "MOOR" returns to "MAP" to continue computation of an output variable using the calculated "OUT OF RANGE" data value as the input value. No extrapolation of the data is performed by "MAP".

If sense line 6 is "HIGH," there is a pending message from a previous call to "MOOR". "MOOR" immediately returns to "MAP" and the program continues as above. Once the user interrogates via the logic pushbutton, the pending message is displayed and control line 9 is reset, which in turn resets the BCD counter. "MOOR" then returns to the user's program.

The only Fortran coding required in the user's program to obtain map out of range messages is a logical declaration and two executable statements. The executable statements are:

```
CALL QRSLL(5,PB5,IER)
IF(PB5) CALL MOOR(0,X,Y)
```

In the first statement QRSLL is a Hybrid Linkage subroutine for testing sense lines. The subroutine arguments are the sense line number (0-5 or 7), a logical variable which will be "TRUE" if the sense line is "HIGH" and "FALSE" if the sense line is "LOW," and an error flag which is required for all linkage routines. In the example above, sense line 5 was used and the logical variable was PB5, which must be declared logical at the beginning of the program.

The second statement is a logical test. If the variable is "TRUE", "MOOR" is called which allows display of the map out of range message and then continues. If the variable is "FALSE", the program continues. In this statement the arguments of the subroutine, "MOOR", must be the integer, zero, and any two scaled fraction variables. these variables are dummy variables that are ignored by "MOOR" if the first argument is zero.

Disk Files (MAPS, MAPLOT)

All of the subprograms, discussed above, are contained on one disk file, named "MAPS".

A program that will provide CRT plots of map data is also available. This program reads the map data identically to the data input subroutine discussed above. The disk file name of

this program is "MAPLOT".

A sample hybrid program follows to illustrate the calls to "MAP", "MAPL", "DATAIN", and "MOOP":

```
C*****SAMPLE HYBRID PROGRAM USING MAP AND MAPL  8/22/78  WMB
C.....INTEGER ARRAYS FOR THREE MAPS
      DIMENSION NA(5),NB(5),NC(5)
C.....SCALED FRACTION ARRAYS FOR THREE MAPS
      SCALED FRACTION FA(33),FB(63),FC(33)
C.....SCALED FRACTION DECLARATION FOR PROBLEM VARIABLES
      SCALED FRACTION XA,YA,XB,YB,XC,YC,ZA,ZB1,ZB2,ZB3,ZC,MAP,MAPL
C.....LOGICAL VARIABLE USED TO DETERMINE WHICH MAP WENT OUT OF RANGE
      LOGICAL PB5
C.....INITIALIZE HYBRID SYSTEM
      CALL QSHYIN(IER,681)
C.....DESELECT ALL CONSOLES
      CALL QSC(0,IER)
C.....SELECT CONSOLE 1 ONLY
      CALL QSC(1,IER)
C.....CALL MAP DATA INPUT ROUTINE
      CALL DATAIN(NA,FA)
C.....READ ADC'S
   10 CALL QRBADS(XA,0,6,IER)
C.....CALL MAP FOR FUNCTION A
      ZA=MAP(NA,FA,XA,YA)
C.....CALL MAP FOR FUNCTION B
      ZB1=MAP(NB,FB,XB,YB)
C.....CALL MAPL FOR SECOND B FUNCTION
      ZB2=MAPL(FB)
C.....CALL MAPL FOR THIRD B FUNCTION
      ZB3=MAPL(FB)
C.....CALL MAP FOR FUNCTION C
      ZC=MAP(NC,FC,XC,YC)
C.....WRITE DAC'S OR DAM'S
      CALL QWBDAS(ZA,0,5,IER)
C.....TRANSFER DAC'S OR DAM'S TO ANALOG CONSOLE
      CALL QSTDA
C.....PULSE PUSHBUTTON 5 "OFF" SWITCH FOR MAP OUT OF RANGE MESSAGE
C.....TEST SENSE LINE 5
      CALL QRSLL(5,PB5,IER)
C.....TEST LOGICAL VARIABLE; IF HIGH, CALL MAP OUT OF RANGE ROUTINE
C     FOR CRT MESSAGE
      IF(PB5) CALL MOOR(0,XA,YA)
C.....RETURN TO READ ADC'S AGAIN
      GO TO 10
      END
```

Statistics

Name: SFDD3, FDDC, FDDR

Type: Fortran callable subroutines

Location: PACER A and B Moving Head Disks

Length: '42

Programmer: David S. Cwynar

Source: Assembly language cards.

Date: June 16, 1978

Subroutines Required: CIPHER I/O library, DISK or NODISK, The Fortran RTL.

Loading Sequence: SFDD3, CIPHER I/O library, DISK or NODISK, the Fortran RTL.

NOTE: MONITOR must reside in core at run time if using the moving head disk (ie. DISK).

Description

These three Fortran callable soubroutines are used to transfer floating point data from the PACER 100 to the IBM 360 or similar computers by producing a $3 type formatted data dump. The device to contain the dump may be paper tape, floppy disk, moving head disk, or any other device acceptable to the AOP subroutine of the CIPHER I/O library.

Use

As of this date, the required CIPHER I/O library disk driver routines (DISK) are not available, however, they will be implemented shortly so a description of their use will be given here.

The first decision to be made, before these routines (SFDD3, FDDC, FDDR) can be used, is how one wishes to format one's data. A single record (ie., length of paper tape, disk file etc.) may consist of one or more data blocks. Each data block of a record must contain the same number of words (ie. the same number of floating point numbers). The purpose of breaking the data record intc blocks is for ease of locating errors when reading the data into the 360. When an error is detected, the message received refers to a line number within a given block. If there are many lines within a block, a lot of line counting could be required when searching the paper tape to correct the error. On the other hand, if blocks are too small, the read-in time will be excessive because of a large number of "overhead" lines which are created for each data block. The user can decide on the data block size to use by studying the $3 format described at the end of this program description. Experience has indicated that blocks between 100 and 200 words are a "nice" size, with up to 500 words not being overly long. The maximum number of data blocks is limited to 100 by the $3 format. The maximum number of words per data block that the PACER 100 can handle is 32767.

Once you have decided on the data block size and number of data blocks/record, you are ready to program calls to the SFDD3, FDDC, and FDDR subroutines into your main program. Your main program will also have to contain the logic to initiate a dump, accept operator commands (such as file names etc.) or any other special operations necessary for controlling the dump process. The sequence of events from the start of a dump to completion of a single record must occur as follows:

1) Initiate the output device to accept data by:

```
CALL FOPENO(IUNIT,NAMEXX)
```

Where: IUNIT is the logical unit number (an integer) of the device to be used (17('21), 18('22), or 19('23) for the moving head disk, 5 for high speed paper tape punch, 20('24) for floppy disk, 16('20) for the line printer, etc.).
NAMEXX is a six character alphanumeric file name. It must be exactly 6 characters long. Any unused characters should be defined as spaces. Failure to provide at least 6 characters will result in unknown characters being picked up and used- hence you won't know the name of your file and may never be able to access it again!

If using floppy disk, the file name is used to indicate the track and sector on which writing or reading will start as well as to select which of two possible disks will be used. Hence, NAMEXX must have the format:

FDXYYZ, where X, YY, and Z indicate disk number (1 or 2), track number (00 thru 63), and sector respectively. The symbols to be used for sectors are listed in Table I. If you wish to manually be able to access the starting sector, such as when reading data into the 360, you must specify sector 00 by making Z the "@" symbol. For example, to write on disk #1 starting at track 55 sector 0, NAMEXX would be as follows: FD155@. To write disk #2 starting at sector 0 track 00, NAMEXX would be: FD200@.

Table I

Symbols for Floppy disk sector numbering

SECTOR #	SYMBOL	SECTOR #	SYMBOL
0	@	16	P
1	A	17	Q
2	B	18	R
3	C	19	S
4	D	20	T
5	E	21	U
6	F	22	V
7	G	23	W
8	H	24	X
9	I	25	Y
10	J	26	Z
11	K	27	RIGHT BRACKET
12	L	28	FORM
13	M	29	LEFT BRACKET
14	N	30	UP-ARROW
15	O	31	UNDERSCROE

To simplify the software and make manual reading possible, there is no directory for the floppy disk. Hence, you must keep accurate records of the starting and ending track/sector for each file. Ending track numbers are easily read from the front panel indicator on the disk unit when all writing to floppy disk has been completed. One need only keep records of track numbers if all files start at sector 0 of the next available track.

The call to FOPENO, as described above, is optional (but recommended) if you are using the paper tape punch. This unit may be readied manually at run time by pressing the "FEED" button before the machine begins its dump. Hence, the program need not call FOPENO.

2) Put the header on the dump by:

```
CALL SFDD3(IUNIT,NPTS,NBLOCK)
```

Where: IUNIT is the logical unit number (an integer) of the device used and must be identical to that supplied to the FOPENO subroutine.
NPTS is an integer = to the number of words per data block.
NBLOCK is an integer = to the number of the last data block in the record.

Once you have completed the above two procedures, the dump is properly started. You must then dump the data by repetitively calling the following subroutines as described below. The value of ICHAN must start at 0.

3) Put the block identification number at the start of a data block by:

```
CALL FDDC(ICHAN)
```

Where: ICHAN = the integer block number.

4) Dump the data by calling FDDR NPTS times (i.e., once for each data word of the block).

```
CALL FDDR(REAL)
```

Where: REAL = a single precision floating point real value to be placed on the dump.

5) Close the data block by:

```
CALL FDDEND
```

6) If another data block remains to be dumped, add one to ICHAN and repeat steps 3 thru 5. That is, steps 3 thru 5 should be performed up to and including the time when ICHAN = NBLOCK.

7) Instruct the output device to empty its buffer and close its file by:

```
CALL FCLOSE(IUNIT)
```

Where: IUNIT = the logical unit number (an integer) of the device being used and must be the same as was used in the call to FOPENO.

NOTE: Step 7 is optional (but recommended) if using paper tape, as the paper tape punch has no buffer to empty and no file to close.

An excerpt from a program using the above subroutines to dump frequency response data from arrays FREQ, XMAG, and PHASE is given below. Here, the number of words per data block (NPTS) and number of the last data block (NBLOCK) is selected by the program from the internal parameters NFREQ and NCHAN. Note that the program decides to initiate a dump by checking sense switch 5. Hence, sense switch 5 serves as a means for operator control. Note also that the program allows the operator the freedom to choose any I/O device and file name.

```
      DIMENSION MYNAME(3)
                .
                .
                .
   57 IF(.NOT.SENSW(5)) GO TO 20
C INPUT UNIT AND NAME FROM OPERATOR
      TYPE 40
   40 FORMAT(3X,32HENTER UNIT NUMBER AND FILE NAME./)
      ACCEPT 50,IUNIT,MYNAME
   50 FORMAT(I5,A6)
C INITIATE OUTPUT DEVICE
      CALL FOPENO(IUNIT,MYNAME)
C START THE DUMP
      CALL SFDD3(IUNIT,NFREQ,2*NCHAN)
C DUMP FREQUENCIES AS FIRST BLOCK (BLOCK #0)
      CALL FDDC(0)
      DO 60 I=1,NFREQ
   60 CALL FDDR(FREQ(I))
      CALL FDDEND
C DUMP MAGNITUDE AND PHASE DATA FOR EACH CHANNEL AS
C   ODD AND EVEN NUMBERED BLOCKS.
      DO 80 I=1,NCHAN
      CALL FDDC(2*I-1)
      DO 70 J=1,NFREQ
   70 CALL FDDR(XMAG(I,J))
      CALL FDDEND
      CALL FDDC(2*I)
      DO 75 J=1,NFREQ
   75 CALL FDDR(PHASE(I,J))
      CALL FDDEND
   80 CONTINUE
C CLOSE THE OUTPUT FILE
      CALL FCLOSE(IUNIT)
C WAIT FOR OPERATOR TO RESET SENSE SWITCH, THEN CONTINUE
   90 IF(.NOT.SENSW(5))GO TO 20
      GO TO 90
   20 XXXXXXX    (Program continues)
```

Format of a $3 Type Formatted Data Dump

1st line The number of points per channel occurs twice in Format 2I6.

2nd line A coded integer appears in format I6. This integer equals the first block number times 100 plus the last block number.

3rd line A repeat of the 2nd line in format I8.

4th thru last lines Blocks of data as follows:

- 1st line-The channel number in I6 Format.
- 2nd line-The channel number in I8 Format.
- next N:---N lines of data followed by a checksum in format 12(I6,I3),I8. The checksum is the sum of the preceeding 12(I6,I3) integer pairs. It represents the sum of 24 integers.
- 2nd last--An odd length line for left over data.
- last line-The checksum for the odd length line. This line does not appear if there is no odd length line. No odd length line will occur if there is no remainder when dividing the number of words per data block by 12.

IBM 029 Card Punch Conversion for PACER (CV29PT)

Statistics

Name: CV29PT

Type: Core-image dump

Location: PACER B moving-head disk.

Program starting address: '1000

Core used: '1000 - '1761 and '0 - '1

Programmer: David S. Cwynar

Source: Assembly language cards

Date: June 28,1978

Subroutines Required: None

Loading Procedure: None

Description

This program reads a deck of cards punched on an IBM029 card punch, converts the symbols to 8 bit non-parity checking ASCII and punches a paper tape suitable for use by the Fortran Compiler or the Assembler. Symbols or multiple-punches which have no ASCII equivalent are punched as rubout ('377).

Use

Only the Core Image dump program is currently available to hybrid users. To use it; 1) Load CV29PT using the EAI software MONITOR. 2) Place cards in card reader and start reader. Note the last card of the deck should be an 8-4 multiple punch in column 1 to put an "@" sign on the end of the deck as is required of all EAI source decks used by the Fortran Compiler and the Assembler. 3) Turn on high speed paper tape punch and manually run out some leader. 4) Start program execution at location '1000 by issuing; #G,1000 in MONITOR.

The program will begin reading cards and punching paper tape. Execution will continue until either the card hopper becomes empty or a card with an 8-4 multiple punch in column 1 is read. If execution stops on the 8-4 multiple punch card, the program will pause, at which time a second deck can be processed by doing: release run-single-run (ie. the program will start over). If termination occurs because the card hopper becomes empty, execution will automatically continue when new cards are placed in the hopper.

To exit the program you must release run, reset I/O and return to the MONITOR. Also, no trailer is punched by the program. Hence, you must manually run out the trailer when your program is complete. Note this operation should be carried out prior to resetting the I/O since, on occasion, an I/O reset causes a noise character to be punched on paper tape.

Interactive Data Collection and Display (INFORM)

INFORM was developed to meet the needs of engineers developing real-time digital programs under time and hardware constraints demanding the use of integer arithmetic and scaled parameters. Initially, INFORM was developed to meet the steady-state data display requirements of such programs. Subsequently, an interactive mode was provided for dynamic display programming as well as debugging and program modification. Sampling and data output subroutines were developed to meet dynamic data collection and real-time data display requirements of transient data.

Reference 5 describes the overall capabilities of the INFORM, DATAO (data output) and SAMPLE (sampling) subroutines and how they are intended to operate within a user's system. Reference 5 also describes their operation. Details on the use of the interactive command structure for accomplishing the possible tasks are presented and examples are given to illustrate the use of the software within a user's system.

The INFORM package is structured into three basic parts; INFORM, DATAO, and SAMPLE. Specifics of the calling sequences and use of each subroutine are given in reference 5. Although they may be called by any main program, they are primarily intended to be executed on the lowest or "spare time" levels of a priority interrupt system, where the main function of the computer, (for example, a process control algorithm), is executed on higher levels. These higher levels are usually driven by recurring external interrupts such as those of a real-time clock or interval timer. When used in this fashion, the command structure made available to the operator by INFORM creates an interactive operating environment for any passive real-time program. Furthermore, the programmer creates this environment by simply defining the interrupt structure and executing a simple "CALL INFORM" statement on the lowest level.

The INFORM subroutine provides an "on-line" display and data manipulation capability for single precision integer values retained in absolute memory. The operator programs the display while the program is running in the interactive mode. Memory locations are referred to by operator assigned, one to five character, alphanumeric names. Values for these named locations may be displayed in engineer's units (EU) or octal. INFORM determines these EU values by multiplying each integer value by a scale factor associated with the name. Once defined, the names and their associated locations and scale factors remain fixed unless re-defined by the operator.

If the system also utilizes the digital computer for data collection, a simple call to SAMPLE where the storing of data would normally be programmed gives the INFORM subroutine interactive control over the data collection process. This gives the operator the option to store any parameter available within the machine at the time of call. The operator also gains the ability to re-structure the available storage. For example 200 time points of 20 variables, 800 time points of 5 variables, or

one time point of 4000 variables are all possible with 4K words of storage. In addition, using the SAMPLE subroutine automatically provides a means for transferring the stored data to a bulk storage device via INFORM's interactive command structure.

In a similar manner, dynamic displays of system variables may be controlled by the INFORM subroutine if the programmer uses the DATAO subroutine for his displays. The programmer need only insure that all parameters to be displayed are in core and that a call to DATAC is executed in his interrupt environment every time he wishes to update the display. All programming is then complete to give the operator interactive control over the display. The operator may then select which parameters are displayed and expand or compress scales at will without concern for scaling or re-calibration of the display recorders. As written, DATAO is intended for use with fixed calibration analog displays using zero to ten volt inputs, but may be modified to work with any display which can be driven to its scale extremities by system digital to analog converters (DAC's).

A complete description of the INFORM program and its use is given in reference 5. It is imperative that any hybrid programmer, desiring to use INFORM, or its supporting routines, EXEC1 and OPSYS1, consult reference 5 before attempting to use them.

The additional information in this manual on INFORM and its use primarily emphasizes differences from and additions to reference 5, incorporated for hybrid users.

Statistics

Name: INFORM (version #2), SAMPLE, DATAO, CLRSMP

Auxiliary entry points/names:

NAVG, EROR, LOC, SPCE, PNAM, CRLF, GSF2, PRFX, CRCM, NAME, SUFX, MATH, CKCR, INPTS, PRFX2, PNUM, UNIT, OPNT, C21, ISF, FIND, SF, ADRS, NAM2, NAM1, GSF, RTRN, IOSET, CMDS

Type: Subroutine object module

Location: PACER A & B moving head disk(MHD) "INFORM" file.

Length: '10467

Absolute core used: '74267 thru '74661

Source program: PACER B "SINFOR" MHD source file.

Programmer: David S. Cwynar

Date: June 1978

Subroutines required:

AIP, AOP, CLOSEI, CLOSEO, CLOSEX, FDDEND, FDD11, FDD22, LOAD, MESAGE, OPENI, OPENIX, OPENIX, OPENO, OPENOX, OUT00, OUT22, OUT77, PUNCH, TTYR, TTYR2, VERIFY, .A22, .C12, .C21, .C27, .C72, .D22, .H22, .H55, .L22, .L55, .N22, .S22

Loading procedure:

Main which calls INFORM, SAMPLE, etc; INFORM, CIPHER, the Fortran RTL.

Description

A complete description of this program is given in reference 5. Auxiliary commands or features which differ from that publication are given below.

Differences

1) Floating point numbers have the unprinted decimal point on the left of the mantissa instead of on the right. That is; 12345 00 is 0.12345 decimal, not 12,345.0.
2) The "space" define/re-define variable name and "/" list core commands have been extended to include an offset parameter as described under "Additional Features" below. This offset parameter (Os) is treated like part of the scale factor and represents the engineering unit value to be applied to a zero value in core. In accordance with this, the scale factor (Sf) retains its original meaning. That is, it still represents the number of engineering units per machine count. This offset parameter is used by the program whenever scaling/descaling is encountered and is also included when calculating DATAO displays so that no changes in operation are apparent to the user. The only exception is the inclusion of an extra operand for the "space" and "/" commands. The new equation used by INFORM to generate engineering unit (EU) values from machine counts (MU) is:

$$EU = MU*Sf + Os$$

Note that if Os = 0.0 no change of operation exists.
3) In keeping with the addition of an offset parameter as part of the scaling, the $1 dump format for SAMPLE collected data has been changed. The second line of a channel's data block now contains two numbers in the Fortran format of 2(I6,I3). The first number is the scale factor as before, the second is the offset and represents the EU value for zero sampled data. The third line of the data block now represents the checksum for both numbers on the second line, and is the algebraic sum of the two I6 values plus the two I3 values. It is still in the Fortran format of I7. Note that if the offset is 0.0, no change in format will be apparent to existing programs using the $1 format.
4) The operator's mode select switch is sense switch H. Setting sense switch H selects the passive mode. Leaving sense switch H reset will select the interactive mode.
5) The numerical value of the default unit numbers used by various commands have been changed to match the PACER unit numbers for the intended device. That is, the line printer is '20 instead of '5 etc.
6) The execution time for SAMPLE, in the non-averaging mode, is approximately 19 microseconds plus 31 microseconds per channel sampled. The averaging mode requires 19 microseconds plus 49.8 microseconds per channel. The timing for CLRSMP is 28.4 microseconds plus 36 microseconds per channel. The time

required to execute DATAO is approximately 8 microseconds plus 51.4 microseconds per channel displayed. INFORM requires 15.2 microseconds per named location to collect the data for a data table printout. A time lag of approximately 52.8 microseconds plus 37.2 microseconds per named location occurs between the entry of subroutine INFORM and the start of data collection.

Additional Commands

In keeping with the command description format of Appendix C of reference 5, the new formats for the "space" and "/" commands are given below. Also given are descriptions for two additional commands- #I and #J.

The #I and #J commands are used to change the command input device. Whenever the command input is changed from the CRT keyboard (unit '2) to some other device such as the card reader (unit '6), the input stream from the new device must terminate with a #I,2 command to restore command input to the keyboard. If an error is encountered in the new device command stream, command input is automatically restored to the keyboard, and reading of the auxiliary device input stream ceases. If the operator wishes to continue input from the auxiliary device without "rewinding" the input file, he should use the #J command, since employing the #I command will reinitialize the input device. If the auxiliary device is the card reader, employing the #I command will force the program to ignore the remainder of the defective card and start with the next card in the hopper. Note, a card reader must be fully buffered and include Hollerith to ASCII conversion in hardware to be useable as an auxiliary input device. Currently only the PACER B system has the required reader. In the following command descriptions, SFAE refers to a sequentially-formed arithmetic expression. References to appendices refer to reference 5.

List core

KEY CHARACTER	OPERANDS
/	{Ad1},{Ad2},{x},{Sf},{Os},{DF11}

Ad1: An SFAE for the starting octal/decimal address. See Appendix B for acceptable formats.
Default value: Last non-defaulted Ad1 used by this command. Initial value=0.

Ad2: An SFAE for the ending octal/decimal address.
Default value: Last non-defaulted Ad2 used by this command if Ad1 was defaulted. Ad2 defaults to Ad1 if Ad1 was entered. Initial value=0.

x: Data/display type. This ASCII character should be "0" for an octal display of integer data, "1" for a decimal

display of integer data, "2" for a floating point display of single precision, floating point real data, "4" for a decimal display of scaled fraction data, "6" for a floating point display of double precision integer data, "7" for a decimal integer display of double precision integer data, and "8" for an ASCII dump of core, the core being interpreted as containing 2 ASCII characters per word.

Default value: Last non-defaulted x used by this command. Initial value=0.

Sf: An SFAE for an octal/decimal scale factor to be applied to the data. See Appendix B for acceptable formats.

Default value: Last non-defaulted Sf used by this command. The initial value assumed is 1.0. Note: This value is ignored for display formats 0, 7, and 8.

Os: An SFAE for an offset value. The offset value is the engineering unit value for a zero value in core. The offset value as currently defined is used as the numerical value for names occurring within the SFAE.

Default value: Last non-defaulted Os used by this command. The initial value assumed is 0.0.

DF11: Octal/decimal number for an output device.

Default value: The operator's console (unit #1).

DESCRIPTION:

Core starting at location Ad1 up to and including Ad2 is displayed on the device selected by DF11. The value for x determines how the data is to be interpreted (integer, floating pt. real, scaled fraction or ASCII), and what kind of display is to be produced. Prior to the display, the data is multiplied by Sf. Sf is ignored for ASCII data.

PROGRAMMER'S FLOWCHART REFERENCE: LIST

Define a name

PREFIX OPERAND	KEY CHARACTER	ADDITIONAL OPERANDS
{Na}	"space"	{Ad},{Sf},{Os}

Na: A one to five character name for a named location. See Appendix B for acceptable formats.
Default value: the no-name parameter.

Ad: An SFAE for an octal/decimal address. See Appendix B for acceptaple formats.
Default value: One greater than the address for that of the last used name operand of previous commands. If Ad is defaulted, the machine will display the default value prior to accepting the Sf entry (see note below).

Sf: An SFAE for an octal/decimal scale factor. See Appendix B for acceptable formats.
Default value: The same scale factor as that of the last used name operand of previous commands.

Os: An SFAE for an offset value. The offset value is the engineering unit value for a zero value in core. The offset value as currently defined is used as the numerical value for names occurring within the SFAE.
Default value: The same offset as that of the last used name operand of previous commands.

Note: If Na is defaulted, the no-name parameter becomes the default name operand for following commands.

DESCRIPTION:

This command is used when it is desired to define a named location. The default option for Na, or the no-name parameter, is especially useful when making program patches or doing random searches during debugging. It allows the operator to manipulate the core with all the power associated with named variables, but without cluttering the defined name list with temporary definitions. The no-name variable is overwritten each time this command is used, but the ensuing definition is used to determine the default values of subsequent commands. That is, the no-name parameter becomes the last used name operand for these commands until a predefined name is used.

PROGRAMMER'S FLOWCHART REFERENCE: DEFN

The following examples illustrate the use of this command for displaying the contents of core and for making program patches. In the examples, the underscores denote system responses and J's denote carriage returns. Also note the use of the = command, which displays the contents of a specified core location, and the " command, which displays the contents of a specified core location and opens that location permitting it to be changed. The use of these commands is explained in detail in reference 5.

To display the contents of core location number 1 using a scale factor of 0. (octal value displayed) with no offset, one would issue the following commands:

```
_"space**1,0,0↵
_=_'000000
```

In this case the no-name parameter was used. To assign the name SPEED to core location number 2 and to display the contents of that location (again, in octal with no offset), we would issue the following commands:

```
_SPEED"space"**,_'000002_↵
_=_'033475
```

In this case Ad was defaulted, therefore, a value of 1 + 1, or 2, became the default location and was displayed by the machine. The default Sf and Os values became zero as previously defined for the no-name parameter.

The following sequence illustrates the use of the INFORM commands to patch machine instructions in locations '400 through '405.

```
_"space"**'400,0,0↵
_"_'000400_was_=_'000000_'22,_now_=_'000022
_"space"**,_'000401_↵
_"_'000401_was_=_'000001_'032404,_now_=_'032404
_"space"**,_'000402_↵
_"_'000402_was_=_'111010_'10405,_now_=_'010405
_"space"**,_'000403_↵
_"space"**,_'000404_↵
_=_'007776
_"space"**,_'000405_1/32768↵
_"_'000405_was_=_'123456_0.5,_now_=_'040000
```

Re-define a name

PREFIX OPERAND	KEY CHARACTER	ADDITIONAL OPERANDS
{Na}	"space"	{xxxxx},{Ad},{Sf},{Os}

Na: A one to five character name for a named location. See Appendix B for acceptable formats.

xxxxx: A one to five character name for a named location to replace name Na. xxxxx may be the same as Na but otherwise must be unique. See Appendix B for acceptable formats.
Default value: The name used for Na (i.e., the name stays the same).

Ad: An SFAE for an octal/decimal address. See Appendix B for acceptable formats.
Default value: The address currently defined for Na (i.e., the address remains unchanged).

SF: An SFAE for an octal/decimal scale factor. See Appendix B for acceptable formats.
Default value: the scale factor currently defined for Na (i.e., Sf will remain unchanged).

Os: An SFAE for an offset value. The offset value is the engineering unit value for zero in core. The offset value as currently defined is used as the numerical value for names occurring within the SFAE.
Default value: The offset currently defined for Na (i.e., Os will remain unchanged).

DESCRIPTION:
This command is used to re-define a named location. All occurrences of variable Na in any tables will now be replaced with variable xxxxx, and the new location and scale factor as defined will be used. The only exception to this rule is when Na occurs in a DATAO table. Here, the new name and location will be used, but because additional values are derived from the scale factor when building DATAO tables, the old scaling for Na will be applied to xxxxx in DATAO outputs. Hence, the DATAO table entry must be re-entered. See Appendix D for commands to revise the DATAO tables.
PROGRAMMER'S FLOWCHART REFERENCE: DEFN(OVLY)

Change the command input device

KEY CHARACTER	OPERANDS
#I	Un

Un: The unit number of the device from which commands are to be obtained.

DESCRIPTION:

This command is used to change the command input stream from the current device to any alternate device. Issuing this command initializes the new device by opening its file. Commands are then read from the new device. The character stream from the device must be ASCII characters which correspond exactly to those characters an operator whould enter for commands being accepted from the operator's keyboard. Note that if the operator's console is not selected, prompt characters which the program normally issues during the acceptance of commands are suppressed (i.e., the ** received for name definitions, etc.). If an error is detected in the command input stream, the command input device automatically reverts back to the operator's console.

PROGRAMMER'S FLOWCHART REFERENCE: SYSTM

KEY CHARACTER	OPERANDS
#J	Un

Un: The unit number of the device from which commands are to be obtained.

DESCRIPTION:

This command is similar to the #I command, with the exception that the input device is not initialized by opening its file. This command is used in place of the #I command if the input device file is already open. This will be the case when the command input was aborted from the new device because an error was detected in the command input stream. Hence, the #J command can be used to restart command input from such aborts by continuing where the input stream left off.

PROGRAMMER'S FLOWCHART REFERENCE: SYSTM

Statistics

Name: EXEC1

Type: Assembly Language main.

Location: PACER moving head disks (MHD).

Length: '1073

Source: Assembly language cards

Programmer: David S. Cwynar

Date: June 16,1978

Subroutines required:

System's programs: (OPSYS1 and OPLIB1) or (OPSYS2 and OPLIB2)

User's programs: BTASK, LOOP, INITAL, LEVEL0

Loading Procedure: Special- see "How to Load Your Program" below.

Note: MONITOR must reside in core if using the moving head disk.

Sense switches:

H- INFORM's mode select switch.

Reset = command mode.

Set = Execute BTASK instead of INFORM as background job.

(Note; you must exit INFORM first using "." command)

G- Set = Do not update the DATAO display.

Analog facilities required: (must all be on same console)

1) 10 KHz pulse train patched into the C1 input of the Real time clock.

2) Control lines 8, 10, 11, 12.

3) External interrupts #0, 3, 7.

Description

This program establishes a time-shared interrupt environment for the Fortran or Assembly language programmer. EXEC1 provides the user with the ability to execute three different programs at three different priority levels. One of these programs gets executed as a background or lowest priority task and may be executed in place of INFORM which is normally run there. EXEC1 also has provision for calling a user's setup program which normally is executed only once prior to setting up and starting the interrupt environment execution. In order to use EXEC1, all programs must be in the form of subroutines so that they may be called at the proper time by EXEC1. The subroutine names the user must use, and restrictions on the programs will be discussed later.

EXEC1 also enables the user to execute the user's program in a main loop while executing INFORM, SAMPLE, and DATAO subroutines at assigned priority levels to produce steady state and dynamic display of scaled fraction and/or integer data which are PACER generated or sampled. INFORM may be used to interactively change the data displays without having to place a hybrid simulation in "HOLD" or otherwise interrupt normal program execution. INFORM also provides program debug features, and allows the user to collect transient data which is present in the PACER 100. This is accomplished through the use of the SAMPLE and DATAO subroutines which are controlled by INFORM. Note that INFORM, DATAO, and SAMPLE are part of OPSYS1 or OPSYS2.

INFORM controls SAMPLE by defining which variables are to be sampled, and how many points for each variable are to be saved. EXEC1 gives control of the sampling process to the user through the use of external clock and trigger signals. The DATAO portion of the program is intended to drive Brush recorders through DAC's/DAM's from data within the PACER 100. INFORM has control over the DATAO display in terms of which variables are displayed and on which DAC's/DAM's they are output from the PACER. Automatic scaling is provided. That is, the operator can select the recorder calibration by specifying engineering unit values for the Brush recorder zero and plus ten volt levels. Proper operation is automatically assured provided the Brush recorders are set up to display zero volts on the right and plus ten volts on the left. The computer can fully compensate for these fixed levels and both positive and negative engineering unit values can be displayed.

Use

1) Segment your program into subroutines. Provisions for descaling and displaying scaled fraction or integer, data will not be necessary since this job is handled by INFORM. Most programs will have three sections: (A) An initilization and setup section (INITAL) which gets executed once prior startup; (B) The main loop (LOOP) which gets updated continually at a fixed rate; and (C) A background task (BTASK) to do things in spare time while LOOP is running. For example, BTASK may change integrator

initial conditions, etc. These sections must be subroutines with names as indicated above and must conform to the restrictions placed upon them as described in the "Creating User Subroutines" section (pg. 69). They should also be programmed in accordance with the considerations listed under "Programming In An Interrupt Environment" (pg. 84).

To use INFORM effectively, INITAL should contain COMMON blocks which define all variables as explained under "NOTES" (pg. 81). Advanced programmers may also wish to create a subroutine called LEVEL0. Novice user's may use the system's dummy LEVEL0 by loading it at the proper time.

2) Compile your subroutines independently. That is, make three independent Object files. Although INITAL may be combined with either LOOP, BTASK, or LEVEL0, we don't recommend it because it can lead to undetected non-reentrant subroutine conflicts which won't automatically get resolved by the standard loading procedure given below.
3) Create a Core Image version of EXEC1 and your subroutines using the procedure outlined under "Creating The Core Image File".
4) Using Monitor, load your Core Image file (Example: #L,MYPROG,21).
5) Using MONITOR, load OPSYS1 (or OPSYS2) which contains INFORM, SAMPLE and DATAO. The command is: #L,OPSYS1,21. The version you load will depend upon whether or not you wish to avail yourself of INFORM's disk file manipulation commands. OPSYS2 is the version being developed for disk manipulation, but note that you must have planned for its use when you created your Core Image file or OPSYS2 may overflow into your COMMON storage areas. Refer to "Creating The Core Image File" (pg. 71).
6) Start execution of your program by issuing #G,1000 from MONITOR.
7) After your initialization subroutine (INITAL) runs to completion, EXEC1 will ask for the loop update time in seconds. If you have a 10 KHz pulse train patched into the C1 input of the real-time clock, you can enter this time directly. For example, enter .010 for a 10 ms. update interval. Your LOOP program will start executing following reception of the carriage return. Therefore, sense switches G and H should be set prior to entering the carriage return. Sense switch G controls the DATAO display. Sense switch H is the INFOPM mode ccntrol switch. The sense switch functions are discussed below and in reference 5. The update time must be at least as long as the longest time it takes your LOOP program to execute. To determine this time, EXEC1 provides timing signals on control lines 10, 11, and 12 which can be used to determine the execution time of each interrupt level. Line 11 goes high when LOOP starts execution, and goes low when LOOP finishes. Lines 10 and 12 do similar things for SAMPLE and DATAO. An oscilloscope plugged into control line 11 on the system console can be used to

obtain the needed information. As a first cut, a long update interval, say 0.1 to 0.5 seconds, may be used to insure that the program will run. You can change this value at any time by stopping the PACER and starting execution over. To avoid executing the INITAL program over such abrupt program stops, start execution at location '1001 instead of '1000 as stated previously.

8) To restart without calling your INITAL subroutine, start at location '1001. To restart without re-initializing the interrupt structure (not recommended), start at location '1002. The programmer can plan for system crashes when programming the INITAL subroutine, by providing a sense switch (not G or H!) for re-initialization of only those items which may need re-setting if the program "bombs". A clever programmer will place latches in common core which can alter the execution of LOOP, BTASK, or LEVEL0. Then, by using INFORM's scale and store command, the operator can dynamically change program execution without ever having to stop running. This feature allows the user to put disturbances into the system to see how controls react, etc. Any parameter listed in COMMON core is subject to INFORM's scale and store operation. Hence, you can quickly change control gains, modify MAP curves, etc., without having to ever stop program execution.

9) The full capabilities and use of the INFORM, SAMPLE, and DATAO package are given in reference 5.

Programming in the Interrupt Environment

Although EXEC1 takes care of most of the complications involved in producing an interrupt environment (such as register saving, establishing level priorities, etc.), the programmer must exercise some care when producing his programs. In a non-stack machine such as the PACER 100, it is very difficult to program reentrant subroutines. What this means is that any subroutines used by a program operating on one priority level may not be used by a program operating on a different priority level. Any software supplied by EAI, such as the Fortran run time library (RTL) or the hybrid linkage routines (LINKN) are either non-reentrant or call non-reentrant routines. Hence, all Fortran compiled programs running on different interrupt levels must have their own copy of the run time library, hybrid linkage routines, or commonly used subroutines. This is easily taken care of by using the #S command of the Core-Image Generator. The procedure is given in "Creating The Core Image File" (pg. 71).

A similar problem of usage by different priority level programs occurs for all input/output operations from the PACER 100 to the analog console, CRT, ADC's, DAC's, line printer etc. The PACER I/O structure demands that all operations be performed in three steps. First, the device is placed in the proper I/O mode, second the device is tested for readiness, and lastly, the I/O operation is performed. During this last step, the mainframe freezes. Therefore, it is imperative that a device be properly set up and ready before I/O is performed.

In general, you should not command similar I/O devices from more than one interrupt level. It is permissable to use the same device on two different levels, however, if its use by the two routines is mutually exclusive (i.e., when routine A is using device X, routine B will not attempt to use X until A has finished). This is usually the case when the operator manually requests dumps or printouts.

For the purpose of determining similar I/O devices, they are grouped as follows:

1) All ADC channels within a system (i.e., both consoles) are considered as one unit.
2) All DAC's within a system unless the "jam specific DAC" ("DO '140-'167" instruction) or QWJDAR and QWJDAS routines are used.
3) Reading/writing any item on either console within a system which involves console selection and/or an analog address, although console selection may be treated as independent from address selection and the read/write operations. That is, it is not necessary that routine A must perform all three functions of console selection, address selection and the actual read/write. It is permissable for routine A to perform console selection exclusively and routine B to select addresses and read/write on the console selected by

routine A.
4) The CRT, keyboard and floppy disk.
5) The line printer
6) The high speed paper tape reader
7) The high speed paper tape punch
8) The moving head disk (units '21 thru '24 all count as one).
9) The card reader.

Since the INFORM program uses several of the above I/O groups, one might assume that one could not employ use of similar devices in one's higher level programs. This need not be the case since INFORM's I/O drivers (the CIPHER I/O library) is designed to be interruptible while awaiting a keyboard input. Hence, no conflict will occur providing that INFORM is "idling" in the command input mode waiting for a keystroke from the operator while the user's program is accessing the common devices. If the operator knows when the user's program is about to access an INFORM device he need merely stop entering command keystrokes until the user's access is complete. This pre-supposes, however, that the operator has sufficient warning so that the processing of any prior INFORM commands which use the device have time to run to completion before the user's program starts.

Creating User Subroutines

Each of the subroutines listed below must be provided. They must be named as indicated, have no arguments except through COMMON, and should perform the functions as described. If you have no need for a particular subroutine, create a dummy subroutine of the proper name which contains the single statement "RETURN".

NAME: INITAL
PURPOSE: To perform those functions which need be executed only once prior to the start of the main loop. It should also contain provision for re-initilization of the program in the event it "bombs" or forces the console into overload. This routine is called only once by EXEC1 prior to starting the loop calculations.
RESTRICTIONS: None, except that INITAL must select a system console and exit with this console selected. The system console is defined as the one on which the 10 KHz clock, control lines, and external interrupts are located.

NAME: LOOP
PURPOSE: This routine is the main calculation loop which gets updated at a fixed update interval as selected by the operator from the allowable field of .0001 to 3.2767 seconds in .0001 second increments.
RESTRICTIONS:
1) LOOP Must not use I/O devices used by INFORM as described under "Programming in the Interrupt

Environment". In any case, the calculation time of LOOP must not expand beyond its allotment or it will cause an update failure. No I/O should be done to slow devices (i.e., anything except DAC's and ADC's). Such I/O should be reserved for subroutine BTASK.

2) LOOP must not select or deselect a console. Furthermore, it should not depend upon a console being selected or deselected to operate. Currently the restriction on console switching is to prevent errors of the real time clock (i.e., the update timer) and the system control line indicators. The clock function is vital and should not be disturbed unless a fluctuating update interval is acceptable. Theoretically, patching the same clock into the Ci input on both consoles (via a digital trunk) should result in minimal disturbance (one count), and thus allow console switching. However, in the future, INFORM may do console selection to read amplifiers, set pots etc., and you may also wish to do this in your BTASK routine. Therefore, it is a good idea to use DAC's and ADC's only in LOOP.
3) LOOP must not use any OPSYS1 routines accessed through OPLIB1.
4) LOOP must also take care of inputting data from the ADC's for the SAMPLE and DATAO subroutines. For details, see "To Produce a DATAO Display" (pg. 73) and "To Collect Sampled Data" (pg. 80).

NAME: BTASK
PURPOSE: To perform auxiliary functions while LOOP is running.
RESTRICTIONS:

1) Considerations must be given to the problems associated with general reentrant subroutines and I/O device sharing with LOOP as described under "Programming in the Interrupt Environment" (pg. 68).
2) BTASK must not enable, disable or otherwise affect any interrupt or interrupt masks, either on the system console or internal to the PACER 100.

NAME: LEVEL0
PURPOSE: To perform operations which require temporarily stopping LOOP from executing. This routine has highest priority and is entered within 70 usec. of when the system console interrupt line 0 is pulsed.
RESTRICTIONS: None, except those listed under "Programming in the Interrupt Environment" (pg. 68).

You must use the Core Image Generator to properly link your compiled programs into an executable Core Image program. Do not use .COP. Starting from MONITOR, the procedure is as follows:

```
#L,CIG,21
#N,MYPROG,21,2
#R,XXXXX,101000,21000
#G,1000
```

where:

MYPROG is a name you assign to your Core Image file, XXXXX is '56430 if you intend to use OPSYS1. Note: XXXXX will be different for future versions of OPSYS. A future version will be required to allow use of INFORM's disk driver commands. You must use INFORM if you wish to manually manipulate disk files, since MONITOR will not run in a time-shared environment. Furthermore, executing MONITOR while OPSYS is in core will destroy the OPSYS input/output buffers. Hence, any pending I/O from INFORM will be lost. The OPSYS program, however, will remain intact and need not be re-loaded.

The following commands are for the Core Image Generator. The asterisk (*) preceeding each command is generated by the Core Image Generator. See reference 2 for a more detailed description of the Core Image Generator commands which follow:

```
*#L,EXEC1,22
*#L,XXXXX,22
```

where:

XXXXX is the file name of your compiled INITAL subroutine. IF your INITAL subroutine calls other subroutines, load them all now by repetitively doing:

```
*#L,YYYYY,22
```

where:

YYYYY is the name of a disk file or files containg the subroutines. NOTE: YYYYY must not contain LOOP or LEVEL0 except under special circumstances.

Follow these loads with

```
*#L,ZZZZZ,22
```

where:

ZZZZZ is the disk file name of your BTASK subroutine. If your BTASK subroutine calls other subroutines, load them all now by repetitively doing: #L,YYYYY,22 as before. Note that YYYYY must not contain LOOP or LEVEL0 except under special circumstances.

Continue with:

```
*#L,OPLIB1,22
*#L,LINKN,22
*#L,RTL,22
```

Now do:

```
*#UL
```

to list any unloaded subroutines. The only names that should show up are LOOP and LEVEL0. If more or less names occur you have either; (1) forgotten to load a YYYYY library, or (2) violated the call conventions listed under "Programming in the Interrupt Environment" (pg. 68). If (1) is your problem, simply load YYYYY and try #UL again. If all is now OK you may proceed. If not, YYYYY probably had to be loaded prior to LINKN, RTL, or some other YYYYY. At this point, the load should be restarted. Do *#M to exit the Core Image Generator. Use .COP to rename MYPROG to JUNK, go back to MONITOR and start over with #L,CIG,21

etc. If (2) is your problem, clean up the disk file by doing: *#M and using .COP to rename MYPROG to JUNK as above. Now go back and read the "Use" and "Programming in the Interrupt Environment" sections.

If the #UL command looks OK you proceed by:

*#S

this is an important step to resolve reentrant subroutine conflicts - don't leave it out!

*#L,AAAAA,22 where:

AAAAA is the disk file name of your LOOP subroutine. If your LOOP subroutine calls other subroutines, load them all now by repetitively doing: #L,YYYYY,22 as before, NOTE: YYYYY must not be SYSLIB. Follow these with;

*#L,LINKN,22

*#L,RTL,22

Now do: *#UL as before. The only name that should appear is LEVEL0. If it doesn't, there is an error. See the preceding paragraph for actions when the #UL listing is incorrect.

Complete the load by:

*#S

*#L,BBBBB,22 where:

BBBBB is the disk file name of your LEVEL0 subroutine. If you don't have a LEVEL0 subroutine, do: #L,LEVEL0,22 and skip the next paragraph.

If your LEVEL0 subroutine calls other subroutines, load them all now by repetitively doing: #L,YYYYY,22 as before. Also load LINKN, and RTL, if needed, as before.

When you are done, the Core Image Generator should type "DN" at the terminal. If it doesn't, there is an error. The corrective action is the same as described previously when the #UL listing is incorrect. If all is OK, finish with:

*#EL

*#M

Your Core Image file is now complete. Check the listing produced by the #EL command to see if you've exceeded the PACER 100 core capacity. MMTOP must be less than COBOT and ZZTOP must be less than '610. If they're not, you will not be able to use INFORM.

An example of a Core Image file formation and listing of the four main subroutines, INITAL, LOOP, BTASK, and LEVEL0, are given in figure 3.

The Core Image file was named F100. After EXEC1 was loaded, the file F100IN was loaded. This file contained the subroutine, INITAL, which selected Console 1 where the 10 KHz clock, control lines, and external interrupts were patched. In addition to linkage and run time library routines, INITAL also called the subroutines, DATAIN, MOOR, and FLIGHT. The file, MAPS, contained DATAIN and MOOR which was loaded next.

At this point the file, FLTCND, containing the subroutine, FLIGHT, could have been loaded. However, FLIGHT was also called by BTASK. Since, in this case at least, non-reentrant subroutine

conflicts between INITAL and BTASK should not occur, the file FLTCND was loaded after the file, F100BT, which contained BTASK. BTASK also called the subroutine, STATE1. The file of the same name was loaded next. It should be pointed out that whether the file, FLTCND, was loaded before or after F100BT made no difference. The overall library requirements would have been satisfied.

OBLIB1, LINKN, and RTL were loaded next. The list of undefined external references was obtained at this point with the #UL command. Since only LEVEL0 and LOOP were missing, the name table was restructured through the #S command.

The file, F100LP, containing the subroutine, LOOP, then was loaded. The only I/O devices that LOOP used were ADC's and DAC's. The only routines, called by LOOP (except for linkage and run time library routines), were MAP and MAPL. Both of these routines were contained in the file, MAPS, which was loaded again.

LINKN and RTL were reloaded to satisfy LOOP's requirements. The #UL command demonstrated that only LEVEL0 was still missing, as it should be. The #S command once again restructured the name table.

The file, F100L0, was loaded to get the subroutine, LEVEL0. The only function of LEVEL0 was to display a map out of range message which required the call to the subroutine, MOOR. Since the file, MAPS, contained this subroutine, it was loaded followed by a third loading of LINKN and RTL.

The Core Image Generator signalled that all requirements were satisfied by typing "DN". The complete memory map listing was obtained with the command #EL. The #M command returned control to the monitor and closed the Core Image File, F100.

DATAO Display

Proceed as follows to produce a DATAO display:

1) The data you wish to display through DATAO must be in the PACER 100 core. Hence, LOOP should sample or perform any calculations needed just prior to exiting. If there is too much data to collect or calculate, stagger it and do only a portion each pass. Better yet, use the DMA (Direct memory access controller, subroutine QRDMAS) which requires only 10 usec/point and allows the PACER 100 to continue running while the data are being sampled. If you do use QRDMAS, however, you must also use QRDMSI to make sure the DMA is finished prior to the first ADC read in LOOP, or else LOOP may generate an I/O conflict on the ADC's by attempting to start a new update before the data being input from the last update is finished. See reference 4 for information on the use of QRDMAS and QRDMSI.
2) Set up the DATAO channel using INFORM's command structure as described in reference 5.
3) Connect the Brush recorders, calibrated to display zero volts

```
#L,CIG,21
CIG CI P2
LD
#N,F100,21,2
FR
F100 CI P1
M_
#R,56430,101000,21000
M_
#G,1000
*#L,EXEC1,22
EXEC1 OB P1
LI
*#L,F100IN,22
F100IN OB P1
LI
*#L,MAPS,22
MAPS OB P2
LI
*#L,F100BT,22
F100BT OB P1
LI
*#L,FLTCND,22
FLTCND OB P1
LI
*#L,STATE1,22
STATE1 OB P1
LI
*#L,OPLIB1,22
OPLIB1 OB P1
LI
*#L,LINKN,22
LINKN OB P2
LI
*#L,RTL,22
RTL OB P2
LI
*#UL
*#S
*#L,F100LP,22
F100LP OB P1
LI
*#L,MAPS,22
MAPS OB P2
LI
*#L,LINKN,22
LINKN OB P2
LI
*#L,RTL,22
RTL OB P2
LI
*#UL
*#S
*#L,F100L0,22
F100L0 OB P1
LI
*#L,MAPS,22
MAPS OB P2
LI
*#L,LINKN,22
LINKN OB P2
LI
*#L,RTL,22
RTL OB P2
DN
*#EL
*#M
M_
```

Figure 3. - Example of core image file formation using the EXEC1 system.

FORTRAN COMPILER REV LEV J00

```
      SUBROUTINE INITAL
      SCALED FRACTION YI(24)
      SCALED FRACTION H,M0,P0,P2,T2,T0,M0RTT0
      SCALED FRACTION F1(322),F2(322),F3(350),F4(518),F5(224),F6(224)
      COMMON/MAPDAT/N1(5),N2(5),N3(5),N4(5),N5(5),N6(5),
     1 F1,F2,F3,F4,F5,F6
      COMMON/FLT/H,M0,P0,P2,T2,T0,M0RTT0
      LOGICAL PB5
      CALL QSC(0,IER)
      CALL QSC(1,IER)
      TYPE 1
    1 FORMAT(/3X,48HPLACE DATA CARDS FOR MAPS AND DAC ICS IN READER.)
      TYPE 2
    2 FORMAT(3X,17HTHEN TYPE RETURN /)
      ACCEPT 3,J
    3 FORMAT(I2)
      CALL DATAIN(N1,F1)
C  ...READ DAC INITIAL VALUES
      READ(6,4)(YI(I),I=1,24)
    4 FORMAT((5(5X,S7)))
C.  . .INITIALIZE DACS
      CALL QWBDAS(YI,0,24,IER)
      CALL QSTDA
      TYPE 5
    5 FORMAT(/3X,42HIGNORE FOLLOWING MAP OUT OF RANGE MESSAGE.)
      CALL MOOR(99, 0S,.0S)
      CALL MOOR(0,.0S,.0S)
C    . PLACE ANALOG IN IC MODE
      TYPE 7
    7 FORMAT(/3X,48HSLAVE CONSOLE 2 TO CONSOLE 1  MANUALLY GO TO IC.)
      TYPE 8
    8 FORMAT(/3X,49HTYPE RETURN TO PROCEED TO DYNAMIC PART OF PROGRAM/)
      ACCEPT 3,J
C ....ESTABLISH ENGINE FLIGHT CONDITIONS
      CALL QRBADS(H,0,2,IER)
      CALL FLIGHT
      DO 10 K=1,16
      KK=K-1
      CALL QWCLL(KK, FALSE.,IER)
   10 CONTINUE
      CALL QRSLL(5,PB5,IER)
      RETURN
      END
```

PROGRAM SIZE = '454

FORTRAN COMPILER REV. LEV. J00

```
      SUBROUTINE BTASK
      LOGICAL SENSW
C.    SET SSW E FOR INITIAL CONDITION UPDATE
      IF(SENSW(5)) CALL STATE1
C     SET SSW D FOR NEW FLIGHT CONDITIONS
      IF(SENSW(4)) CALL FLIGHT
      RETURN
      END
```

PROGRAM SIZE = '17

FORTRAN COMPILER REV LEV. J00

```
      SUBROUTINE LOOP
C ....FLIGHT CONDITION VARIABLES
      SCALED FRACTION P0,P2,T2,T0,M0RTT0
C.....MAP DATA TABLES
      SCALED FRACTION F1(322),F2(322),F3(350),F4(518),F5(224),F6(224)
C ....ADC VARIABLES
      SCALED FRACTION X0,X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,
     1 X14,X15,X16,X17,X18,X19,X20,X21,X22,X23
C.... DAC VARIABLES
      SCALED FRACTION Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8,Y9,Y10,Y11,Y12
C . ..OTHER VARIABLES
      SCALED FRACTION MAP,MAPL,V4,V7,V8,V9,V10,REY,AR,PRSUB,DPR,PE,
     1 PRE,PRSUP,FRD,RT4,RT41,AE,A,DY7,PRC1,PRC2,Y6SQ,Y7SQ
      COMMON/OUTVAR/X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,
     1 X15,X16,X17,X18,X19,X20,X21,X22,X23,Y0,Y1,Y2,Y3,Y4,Y5,Y6,
     2 Y7,Y8,Y9,Y10,Y11,Y12,V7,V8,V9,V10,REY
      COMMON/MAPDAT/N1(5),N2(5),N3(5),N4(5),N5(5),N6(5),
     1 F1,F2,F3,F4,F5,F6
      COMMON/FLT/X0,X1,P0,P2,T2,T0,M0RTT0
   20 CALL QRBADS(X2,2,3,IER)
      Y7=MAP(N4,F4,X3,X4)
      Y5=MAPL(F4)
      IF(X2.LT..0S) X2=.0S
      A=MAP(N1,F1,X2,X4)
      Y7=(Y7*(.5S+.5S*A))/.5S
      DY7=.0S
      REY=(.56715S*P2*(.9S*T2+.17874S))/(T2*T2)
      IF(REY.LT..30584S) DY7=-.02626S*REY+.00803S
      IF(REY.LT..14202S) DY7=-.15077S*REY+.02571S
      Y7=Y7-DY7
      Y3=Y7
      CALL QWJDAS(Y3,3,IER)
      CALL QWJDAS(Y5,5,IER)
      CALL QWJDAS(Y7,7,IER)
      CALL QRBADS(X5,5,3,IER)
      IF(X5.LT..0S) X5=.0S
      V4=MAP(N2,F2,X5,X6)
      Y6=MAP(N3,F3,X7,X6)
      Y6=(Y6*(.5S+.5S*V4))/.5S
      Y2=Y6
      CALL QWJDAS(Y2,2,IER)
      CALL QWJDAS(Y6,6,IER)
      CALL QRBADS(X8,8,3,IER)
      RT4=SSQRT(X8)
      V7=(.79567S*X10)/RT4
      Y8=MAP(N5,F5,X9,V7)
      V9=MAPL(F5)
      Y9=(V9*RT4)/.78259S
      CALL QWJDAS(Y8,8,IER)
      CALL QWJDAS(Y9,9,IER)
      CALL QRBADS(X11,11,3,IER)
      RT41=SSQRT(X12)
      V8=(.82695S*X11)/RT41
      Y10=MAP(N6,F6,X13,V8)
```

```
      V10=MAPL(F6)
      Y11=(V10*RT41)/ 46715S
      CALL QWJDAS(Y10,10,IER)
      CALL QWJDAS(Y11,11,IER)
      CALL QRBADS(X22,22,2,IER)
      Y4=X22*X23
      CALL QWJDAS(Y4,4,IER)
      CALL QRBADS(X14,14,5,IER)
      IF(X1.GT .22000S) GO TO 21
      AE=.04162S*X14-.08124S+.73465S*X17
      GO TO 22
   21 AE= 02585S*X14-.05837S+.80875S*X17
   22 AR=AE/X17
      DPR=(.69683S*AR*AR-AR+.32094S)/.10567S
      PRSUB=.52828S-DPR
      PE=PRSUB*X18
      FRD=X16*M0RTT0/.79237S
      IF(.20000S*P0.LT.PE) GO TO 23
      PRE=.20000S*P0/X18
      Y12=((X15*SSQRT((.2578S-.2578S*PRE)*X14))/.7652S-FRD)/.34933S
      GO TO 24
   23 PRSUP=.52828S+DPR
      PE=PRSUP*X18
      Y12=((X15*SSQRT((.47303S-PRSUP/.90106S+ 85065S*PRSUP*PRSUP)
     1 *X14))/.69633S-FRD-(AE*( 20000S*P0-PE))/.4293S)/.34933S
   24 Y7SQ=Y7*Y7
      Y0= 87889S*Y7SQ-.00605S*Y7+.23058S
      Y0=(Y0-X3)/Y0
      Y6SQ=Y6*Y6
      PRC1=.92207S*Y6SQ-.26212S*Y6+.21015S
      PRC2=.11866S*Y6SQ- 14705S+.91948S*Y6
      Y1=( 93333S*PRC1*(.99999S-X5)+X5*PRC2)/.93333S
      Y1=(Y1-.90121S*X7)/Y1
      CALL QWJDAS(Y12,12,IER)
      CALL QWJDAS(Y0,0,IER)
      CALL QWJDAS(Y1,1,IER)
      CALL QRBADS(X0,0,2,IER)
      CALL QRBADS(X19,19,3,IER)
      RETURN
      END

PROGRAM SIZE = '1301
```

```
PAGE  1      C*****F100 INTERRUPT ROUTINE FOR MOOR  10/12/78  WMB

FORTRAN COMPILER REV. LEV. J00


      SUBROUTINE LEVEL0
      LOGICAL PB5
C  ...PULSE PB5 FOR MAP OUT OF RANGE MESSAGE
      CALL QRSLL(5,PB5,IER)
C.... DUMMY CALL TO MAP OUT OF RANGE ROUTINE
      IF(PB5) CALL MOOR(0,.05,.05)
      RETURN
      END


PROGRAM SIZE = '26
```

on the right and plus ten volts on the left (red buttons depressed), to the selected DAC's.

4) DATAO will update as quickly as possible for the time available and will attempt to produce an update for each pass through LOOP. If insufficient time remains to update all channels, no time will be left over for INFORM and its operating speed will be extremely slow. To get back to INFORM, depress sense switch G which stops the DATAO updates. To resume the updates, release sense switch G. An alternative is to increase the update time by stopping the PACER and restarting at '1001.

Data Sampling

Proceed as follows to sample and store PACER data:

1) Patch a clock into external interrupt line 3 which goes high at the rate you wish to collect samples. It is not necessary to turn this clock off when not collecting samples, as EXEC1 performs this operation automatically by ignoring the interrupt.
2) Set up SAMPLE for data collection by issuing the proper INFORM commands as described in reference 5. Don't forget to set the block size and averaging mode by using the "B" command first.
3) Pulse interrupt line 7 to start a reading.
4) The data to be sampled must be in the PACER 100 core. This is a job for your LOOP or LEVEL0 subroutines. Which you use will depend on the priority of your samples. If you do them in LOOP (as is the usual case) the samples will be synchronized with your LOOP update, the start of each new sample starting immediately following the first complete update interval which follows the clock rise on interrupt line 3. EXEC1 provides a logical variable called STIME which goes "TRUE" for the one update pass through LOOP which will precede the taking of the first sample. Hence, STIME can be tested by LOOP to determine if ADC's should be commanded to sample variables not already in core. To gain access to STIME by a Fortran LOOP program, declare STIME as logical and define STIME as being location '670. The statements are:

```
LOGICAL STIME
DEFINE STIME('670)
```

You need not set STIME false after the samples are taken since EXEC1 will do this as soon as you leave LOOP. You can ignore STIME altogether if your samples are always in core at the completion of each pass of LOOP.

5) After the first sample is taken, EXEC1 will raise control line 10. Control line 10 will stay high until the last sample of the first entire transient is in. Hence, it makes a good step disturbance generator for starting transients to be sampled. EXEC1 detects when SAMPLE has been placed in the averaging mode by the operator, and increments the average number accordingly. If no averaging is being used (ie. only a single transient instead of multiple identical transients is being sampled per run), EXEC1 will automatically turn off the

sampling process at this point and the operator can dump out the samples using one of the INFORM commands.

If the averaging mode has been selected, control line 10 will go low as soon as the first transient is complete, and a new transient will start on the next rise of the clock on interrupt line 3. If the simulation needs time to settle out between transients, the user must use the logic area of the 680 analog console to detect the falling edge of control line 10 and use it to hold the clock connected to interrupt line 3 low until the simulation is ready to go again. This process will repeat until the number of averages specified by the operator through the "B" INFORM command have been taken. Don't forget that the averaging mode may also be used to filter noise from steady state data by specifing the number of samples as 1, and asking for, say, 1024 averages. If this is your mode of operation, you can forget about the rise and fall of control line 10 and just keep the clock going into interrupt line 3. Don't set the clock rate too high, though, as you may choke the system by allowing insufficient time to collect samples. You can look at control lines 10 and 11 on a scope to see how fast you can go. Also, for good noise filtering, the samples should be spaced out over several seconds.

If a situation occurs where the interrupt structure loses its integrity, the message; "SYSTEM CRASH" followed by an appropriate message will occur on the CRT terminal. To recover from such an error you must re-initalize the interrupt as described in paragraph 8 of the "Use" section. If INFORM is operating in the variable trip mode with the halt option selected, the message "INTERRUPTS ABORTED!" will be received when the trip occurs. This means that all interrupt processing has stopped and that any interrupt levels which were in progress when trip occurred have been left and shall not be completed. The only routine running after receipt of this message will be the INFORM or BTASK subroutine as selected by the state of sense switch H immediately following the printout of this message. The situation is temporary, however, and normal execution will resume when INFORM or BTASK are exited (use the INFORM "." command). The above message and operation will also occur any time the PACER executes a trap instruction of number 0 thru 10. Hence, BTASK can halt program execution at any time by executing a TRAP 0 instruction.

Notes

1) At present, INFORM, SAMPLE, and DATAO can only manipulate scaled fraction or integer data, although INFORM's List Core command can display any data type on a one at a time or one block at a time basis.
2) You must know the core location (by number) of all data you wish to manipulate. Hence, you should place variables in COMMON core. Place all inputs/outputs, map inputs/outputs and any key intermediate calculations there since you never know what may be useful to observe when

debugging a malfunctioning program.

3) Building the definition table for a new program, although no more difficult than typing up a program, is not easy. Hence, you only want to do it once and you only want to make changes to it when your program gets updated. You can accomplish this through the use of INFORM's relocatable load/dump feature if your assigned COMMON locations remain fixed. In the PACER, COMMON is assigned from the top of core down. The last-defined COMMON block of the first loaded program occurs first. COMMON statements not contained in the first loaded program are appended to the table from the top down. The COMMON which is defined last in the new program, appears first or at the first core location below the first COMMON blocks loaded. Therefore, to be able to add new variables to COMMON as your program expands or changes without having all your COMMON shift because you increased or decreased the length of a COMMON block, your first program loaded should contain <u>all</u> your COMMON statements. You may then add COMMON to any subroutine or portion of your program by adding variables to the <u>first and only the first</u> COMMON block defined by this program. Alternately, you may add a new COMMON block definition to the front (i.e., preceding any existing COMMON block definition) of this "master" program. If you wish to keep your COMMON blocks segmented into nice blocks, it might not be a bad idea to build expansion room into each block by adding several dummy locations to each COMMON block. It's easy enough. Just add a dummy dimensioned variable to the end of the COMMON. You should still follow the procedure outlined above, however, just in case you run out of expansion room.

The priority granted each routine from highest to lowest is as follows:

LEVEL	SIGNAL TO CAUSE INTERRUPT	ROUTINE CALLED
0	EXTERNAL #0	LEVEL0
1	REAL TIME CLOCK	LOOP
2	----------	------
3	----------	------
4	EXTERNAL #3	SAMPLE
5	----------	------
6	----------	------
7	NONE(LOOP completion)	DATAC
8	----------	------
9	EXTERNAL #7	CLRSMP
10	----------	INFORM or BTASK

Note that the level 9 interrupt calls CLRSMP, which is an auxiliary subroutine provided in INFORM for clearing all SAMPLE storage blocks. It is also used to initiate the summation in the averaging mode. Details of the INFORM operations and options are provided in reference 5.

Indicators and Operating Features

1) Pulse interrupt line 7 to clear the SAMPLE core and start a new reading.
2) Pulse interrupt line 3 at least once for each sample point (i.e., the sample clock goes here). This clock may be continuous as the program uses pulses only when sampling.
3) Control line 8 is an update failure indicator.
4) Control line 10 comes high after the first sample if a reading is taken and goes low when the reading of the first block is complete. If SAMPLE is operating in the averaging mode, this line will go low once for each average taken.
5) Control line 11 goes high when a new update through LOOP starts, and goes low when the pass is complete.
6) Control line 12 goes high when DATAO starts a display update, and goes low when complete.

Statistics:

Name: OPSYS1

Type: Core Image dump less main.

Location: PACER A and B moving head disk (MHD).

Core used: '56433 - '73521 and '466 - '616

Programmer: David S. Cwynar

Source: None

Date: August 1978

Loading sequence: NONE

Description:

OPSYS1 is a Core Image load of INFORM (version #1) and all its required subroutines.

Use:

The sole purpose of OPSYS1 is to save the user of INFORM a lengthy loading procedure and to shorten the length of Core Image modules which use INFORM by having all such modules share a common INFORM load- namely OPSYS1.

The load sheet used to produce OPSYS1 is given in figure 4. It shows the entry address of all the CIPHER I/O library routines and external references available from this load. These external names and addresses are duplicated in an Object file called OPLIB1 where each name is equated to its corresponding location in the OPSYS1 dump. Hence, a user's program can avail itself of any subroutine or external name resident in OPSYS1 by simply programming in the same manner as he would for any normal subroutine or external name and loading OPLIB1 instead of the subroutine when creating his Core Image file. OPLIB1 will generate the address required by the Core Image Generator to satisfy the user's call or external statement. Note that the routine which is present in OPSYS1 will not become part of the user's Core Image file, but the user's program will execute a part of OPSYS1 whenever a call to an OPSYS1 subroutine is encountered. Hence, when the user wishes to run his program, he must load OPSYS1 in addition to loading his Core Image file. Note further that his Core Image file must not infringe on locations used by OPSYS1 as listed above unless the user is sure

that the portion of OPSYS1 he overlays will not be required by his program.

OPSYS1 contains the version 2 INFORM Object module which does not include the auxiliary # commands used for controlling the moving head disk. Also, NODISK was loaded in place of DISK to produce an error message whenever attempts to read/write any of the disk units is attempted.

Note that since INFORM is strictly a subroutine, the OPSYS1 file is <u>not</u> a complete program since it is missing a main program to call INFORM. INFORM, and hence OPSYS1, may be treated as a complete program which requires no main for execution providing that it is run strictly in the interactive mode. Therefore, OPSYS1 may be run as a main program provided that the operator's mode select switch(sense switch H) is set permanently to interactive mode (reset) and providing that the "." INFORM command to exit INFORM is never used. The return address for the "." command has been set to produce an error if the "." command is issued when executing INFORM (or OPSYS1) strictly as a main program. To operate OPSYS1 as a main program, the starting address is that as listed for CMDS on the load sheet.

```
@X,2
.COP CI P1
LD
C_
!$IN,22,INFORM
!$FILE,22,BSS68,F
!$FILE,22,CIPHER
!$FILE,22,NODISK
!$LOAD,S56433,Z466,C77777
FR
FR
CRMGAA CI P1
CIG CI P2
LD
INFORM OB P1
LI
BSS68 OB P1
LI
CIPHER OB P1
LI
NODISK OB P1
LI
RTL OB P2
DN
.COP CI P1
LD
C_
!$RENAME,21,CRMGAA,OPSYS1
CRMGAA CI P1
FR
CRMGAA C1 P1
OPSYS1 CI P1
!$ASM
FR
FR
```

```
OBJTAA OB P1
ASSEM C1 P2
LD
FR
 COP C1 P1
LD
C_
!$RENAME,22,OBJTAA,OPLIB1
OBJTAA OB P1
FR
OBJTAA OB P1
OPLIB1 OB P1
!
```

Figure 4. - OPSYS1 load sheet.

REFERENCES

1. 681 Parallel Analog Processor Refence Handbook. Publ. No. 00800.2074-0, Electronic Associates Inc., Jan. 1974.

2. PACER 100 Digital Computing System Moving Head Disk Software System Manual. Publ. No. 00827.0061-1, Electronic Associates Inc., May 1975.

3. PACER 100 Digital Computing System Library and Utility Manual. Publ. No. 00827.0052-4, Electronic Associates Inc., Apr. 1975.

4. PACER 100 Digital Computing System Applications Programming Manual. Publ. No. 00827.0051-3, RD-1, Electronic Associates Inc., Oct. 1977.

5. Cwynar, David S.: INFORM: An Interactive Data Collection and Display Program with Debug Capability. NASA TP-1424, 1979.

End of Document