VOL. 7, NO. 2 APRIL/JUNE 1984

JOURNAL

INTEREX. THE INTERNATIONAL ASSOCIATION
OF HEWLETT-PACKARD COMPUTER USERS

JOURNAL
JOURNAL
JOURNAL
JOURNAL
JOURNAL
JOURNAL
JOURNAL
JOURNAL

JOURNAL

CONTENTS

An HP 1000 Series building equipment preventative maintenance program 2
Harry McLean

Testing compares new half-inch tape drive to existing drives 9
Bob Gilbert

Pascal strings and MPE 13
Joseph Berry

Introduction to systems programming on the HP 3000 using Pascal 16
Stephen Tucker

EDITOR'S NOTE

As usual, the Publications Committee is grateful to the authors for the quality of their efforts, as evidenced in this issue. Your *Journal* is a major vehicle for sharing information about what works and what needs work.

We are especially pleased to have an article featuring an HP 1000 application. This represents a first since the recent merger of the Users Groups.

Keep the material coming!

—John Ray

INTEREX, 2570 EL CAMINO REAL WEST, FOURTH FLOOR, MOUNTAIN VIEW, CALIFORNIA 94040, (415) 941-9960

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

JOURNAL

An HP 1000 Series building equipment preventative maintenance program

Harry McLean

Hewlett-Packard
Vancouver, Washington
USA

ABSTRACT

Preventative maintenance (PM) is a critical program for any modern facility. Unscheduled downtime can be very costly and has a great impact on operational efficiency. Following is a program that will produce a written PM procedure for each piece of equipment in the month it is scheduled for preventative maintenance. It is written in the terminology familiar to maintenance personnel.

COMPUTERIZED PM

Although potentially a valuable tool, the computer has been overlooked in many building maintenance departments. Software programs presently available can automate and enhance the productivity of the building maintenance department. One of them is a Preventative Maintenance (PM) program developed at the Vancouver Division of Hewlett-Packard. Various modules in this program "massage" PM information and present data for both the repair person and the department manager.

One of the obstacles in implementing a computerized PM program is the reluctance or resistance of personnel who don't use computers. Nonusers often see the computer as a threat to their livelihood. This is a real problem, and one of the ways to deal with it is to make sure the user accesses programs that are friendly and forgiving. In other words, by using messages and terms that are not foreign to the user, the program will be perceived as a helpmate.

The entire PM system operates on a Hewlett-Packard 1000 computer system. The system requires either the RTE-6V/M or RTE-IV operating system, with FORTRAN 4 or FORTRAN 7, and the HP Assembler or Macro. Figure 1 lists the hardware required to implement this system, although it can call for additional hardware for specialized applications—facilities energy management, workorder and project management, project management, weather data analysis, and so on. Now to the PM program.

All of the programs can be activated after logon by pressing the appropriate softkey. From this point on, all of the programs are driven via menus that are displayed on the user's terminal. If a numeric input is expected for a response and an alpha input is typed, a friendly error message will be displayed. The user can then retype the correct value and continue, without aborting the program or displaying an error message that has no significance to the user. The various listings are all produced on 80-column paper.

There are eight programs that can be activated by pressing the appropriate softkeys. Six of these are available to the general user, and the other two are available to department managers as a tool for better managing each group's labor hours. The six user programs follow:

1. The main PM program has the following capabilities. It can:
   - add a device to the system database
   - modify a device on the system database
   - delete a device on the system database
   - clear the PM due flag.
   - list devices that are on the system database
   - add or modify repair histories.
   - update PM labor and material costs
   - update calibration times.

   (A device is an item, such as a return fan, for which PM is deemed necessary).
2. The PM codes program can do the following:
   - add a PM code
   - modify a PM code (code description or number)
   - list the PM codes.
3. The PM check program allows the users and managers to check the progress of the PM during the month. After the appropriate softkey is depressed, the program displays a message and searches the PM database "due or overdue" devices. The results, by group, are then displayed and totaled. The user can then opt to list the devices that are due or overdue on the system printer or terminal or to exit the program.
4. After accumulating repair histories, the user or manager can selectively list—in various formats—the repair records for a particular device and the repair records for all devices, sorted by device number or sorted by repair record number. This program can help to determine if a device should be obsoleted, or if the PM schedule should be revised.
5. Repair histories can contain as many as four failures in each repair record. These can be combined as mechanical, electrical, or other types of failures on the same repair record. The program that creates or adds failure codes also allows for the changing of a failure code or the listing of the failure codes.

6. In the event that someone forgets to set the line paper to top-of-form or loses the PM report, all is not lost. This program, when run, will produce the entire listing of PMs due for the month, either in its entirety or by device number.

The PM program automatically searches the device database for all of the devices that are due for preventative maintenence on the first day of every month. If the device is due, a PM report is listed (see Figure 2), and a software flag is set for that device. With this report the entire PM can be performed on the device within the current month. If the person either doesn't perform the PM or forgets to clear the PM flag (see the fourth item under the main PM program), this device will be listed, and a message will be printed that it is overdue on all subsequent months until its flag is cleared.

Included in the monthly PM report for each device are many valuable bits of information, such as:

- the entire device specifications;
- year-to-date PM labor hours and PM material costs;
- date PM last performed and by whom;
- repair history summary, i.e., number of mechanical, electrical, or other failures, total repair material costs, and labor hours. Also, if any, all repair history record numbers are listed;
- comments: a 10-line by 70-charcters/line "scratchpad." The user may include information such as additional precautions to take during the PM or use it as a mindjogger. Each line can be modified independently by the user, and each device has its own comment field;
- the PM operation codes associated with the particular PM interval, with their descriptions, i.e., PM code 10 could be "Verify oil level and fill if necessary." The PM standard labor hours for each operation/code are also listed.

Each device must be stored on the PM database via the main PM program. If any particular information is lacking or incorrect at the time of storage, it can be modified or added at a later time. There are six different PM intervals, and any combination may appear on one device. They are:

1. monthly
2. bimonthly (every other month)
3. quarterly
4. semiannually
5. every nine months
6. annually.

If a device is to have monthly as well as quarterly PMs, the quarterly PM codes will be printed on the PM report on the month when both the month and quarterly PMs are due. In other words, the monthly PMs will be listed by the PM scheduling program every month until a quarterly PM is due. For that month, the PM report will contain operations and codes that are unique to a quarterly PM. This holds true for any PM when a monthly and some other PM frequency are due simultaneously.

In order to delete a device and/or its repair histories from the databases, a special password is needed. Even after the correct password has been typed in, the computer will ask if the user is sure he or she wants to delete that device! This feature is important because the user may inadvertantly enter a delete routine and wish to exit without destroying any PM data.

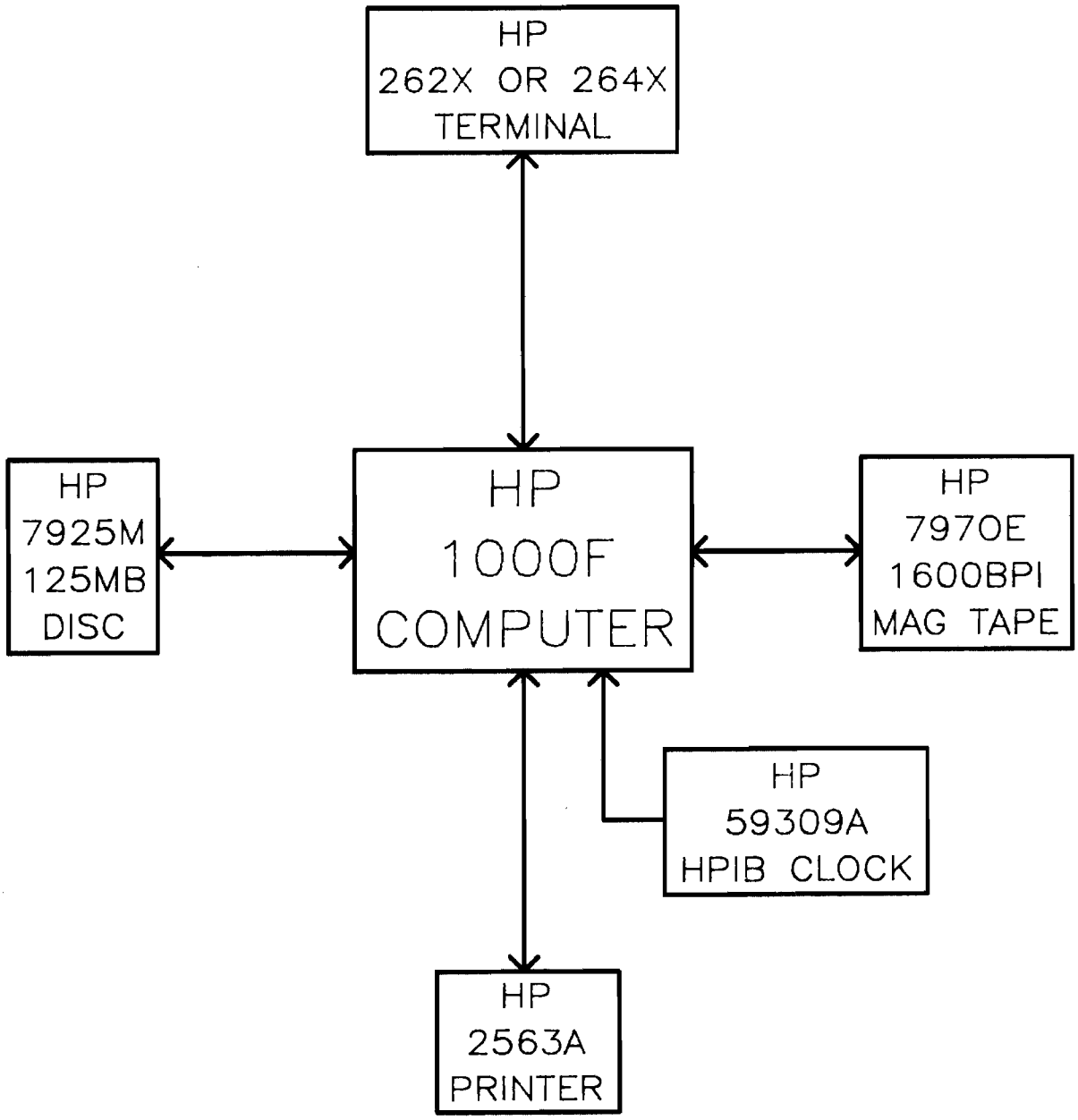
Listing devices to the system printer can be done through the main PM program. The choices are to list all devices, list one device, or list a block of devices. All of these listings will include all the device specifications, comments, and, if desired, the repair history summary. An additional listing (Quick-list) can obtain a list of all of the devices on the PM database with their device number, identifier, group number, location, and building number (see Figure 3).

The two programs available only to the supervisor are the *Workload Distribution Analysis* and *Cost and Labor Totals* for each device. The *Workload Distribution Analysis* report is originally run by the monthly PM scheduling program. During the search and listing of the devices, the PM standard labor hours for each device is accumulated by maintenance group (see Figure 4). This summary includes—for each maintenance group—the total number of devices due and overdue and their total standard hours, by PM priority and zeros in all the columns (repair, workorder, overtime hours), except the net hours, which will have a negative of the standard hours. The supervisor next needs to determine the workorder, regular, and overtime hours for each group and to save this information via the *Workload* program. The program will then calculate the correct net hours and list each maintenance group's workload schedule. As device repair histories are added to the PM system and workorder hours become known, the *Workload* program can be run as often as necessary. All of the repair hours by maintenance group will then be accumulated and the new net hours calculated for each group. On the first of every month, the workload listings for each group are listed according to the last available information, the accumulators cleared, the device PM reports produced, and then the current month's workload listings are produced. Each supervisor can assign a unique password to each of the group's workload files for security measures.

The *Cost and Labor Totals* program (Figure 5) will produce a listing of all of the devices on the PM database with their descriptions, the date the device was entered on the system, and labor hours for repair, with year-to-date and prior PM labor hours and material costs for the same. This listing can be of value in determining where the PM labor hours are being spent. The prior PM labor hours and prior PM material costs are values accumulated from times before the start of the current year.

This PM program has been used by the technical maintenance group (electrical, plumbing, electronics, industrial,

HVAC) at Hewlett-Packard Vancouver Division since August, 1982. Comments have been favorable and have supported the program: Enhancements and changes to the PM programs have been implemented as a result of suggestions from this group. Recently, the general maintenance group (auto maintenance, landscaping, and general maintenance) also started using the system. Ten other HP divisions currently are using this system, and another is in the process of purchasing a computer that will allow it to implement the Preventive Maintenance system.



Hardware for Preventative Maintenance System

Figure 1

PREVENTATIVE MAINTENANCE MONTHLY FILE LISTING PRIORITY: C DATE RUN 02/01/84 ON SYSTEM DATE : 882
LAST REV DATE :1083

* LOG #	IDENTIFIER	TYPE	ASSET #	PHYS LOC	BLD	GPR	LOC*	*FREQ	START	PM YTD LAB HRS	PM YTD NTL COST	PM YTD DONE	TECH*
418	HOT WATER PUMP 7	PUMP MOTR	ENL/T7	1	3	349140*	4	883.		0.00	0.00	184.	LS*

MOTOR SPECIFICATIONS

MFG	RATINGS	FLA	MODEL #	SERIAL #
G.E.	Z/460/3	2.9	145TY	3K145DLZ266A

PRIOR PM * LAB HRS= 1.22 NTL COST= 0.00*

PUMP SPECIFICATIONS

MFG	MODEL #	SERIAL #	COUPLING	PRESSURE IN	PRESSURE OUT
PACO	SEE COMMENTS	B2WATBP79117	DIRECT	16	42

* PMCODE #	PM STD	DESCRIPTION
8	.050	CHECK MOTOR FOR VIBRATION
9	.050	CHECK/RECORD MOTOR AMPS _____.__AMPS
74	.050	CHECK BEARINGS FOR HEAT AND WEAR AND NOISE
75	.050	RECORD INLET P.S.I. (LOOK FOR EXTREAMS)____P.S.I.
76	0.000	RECORD EXAUST P.S.I.(LOOK FOR EXTREAMS)____P.S.I.
39	0.000	CHECK SEALS FOR LEAKS
31	.050	GREASE MOTOR BEARINGS AS PER PM SCHEDULE

Total PM hours= .250

* COMMENTS OR NOTES :

PUMP MODEL CAT#16209557301011562

* REPAIR STATISTICS : LABOR HRS= 0.00 NTL COST= 0.00 # REPAIRS= 0

NUMBER OF ELECTRICAL REPAIRS=	0	MATERIAL COST, $	0.00	LABOR HOURS	0.00
NUMBER OF MECHANICAL REPAIRS=	0	MATERIAL COST, $	0.00	LABOR HOURS	0.00
NUMBER OF OTHER REPAIRS=	0	MATERIAL COST, $	0.00	LABOR HOURS	0.00

* REPAIR HISTORY RECORD NUMBERS :

********** THIS DEVICE IS OVERDUE FOR PM **********

Figure 2

PREVENTATIVE MAINTENANCE ABBREVIATED FILE LISTING DATE : 03/21/84

LOG #	IDENTIFIER	TYPE	LOCATION	GROUP	BLDG #
1	HAND TAPER	3	UL/TPM	2	1
2	TOUCH TAPER 1	3	SHIPPING	2	1
3	TOUCH TAPER 2	3	SHIPPING	2	1
4	TOUCH TAPER 3	3	SHIPPING	2	1
5	TOUCH TAPER 4	3	S.PARTS	2	1
6	TOUCH TAPER 5	3	KITTING	2	1
7	TOUCH TAPER 7	3	SHIPPING	2	1
14	DOMISTIC HOT WATER PUMP 1	2	WML/B3	2	1
15	DOMESTIC HOT WATER PUMP 2	2	EML/U9	2	1
16	DOMESTIC HOT WATER PUMP 3	2	EML/U4	2	1
17	DOMESTIC HOT WATER PUMP 4	2	EML/U4	2	
18	DOMESTIC HOT WATER PUMP 5	2	WML/C9	2	1
19	IRRIGATION PUMP	2	COMMENTS	2	1
20	SUMP PUMP (PORTABLE)	3	LS 109	2	1
39	AIR RECIVER TANK	3	BASEMENT	2	1
40	COMPACTOR	3	OUTSIDE RC	2	1
41	TORK LIFT	3	OUTSIDE RC	2	1
42	HANKINSON AIR DRYER	3	BASEMENT	2	1
43	ATLASCOPCO COMPRESSOR 2	3	BASEMENT	2	1
44	INGERSOLL/RAND COMPRESSOR 1	3	BASEMENT	2	1
45	HOUSE VACUUM	3	BASEMENT	2	1
46	VACUUM PUMP 1	3	BASEMENT	2	1
47	LIQUID RING VACUUM PUMP 2	3	BASEMENT	2	1
48	EMERGENCY GENERATOR	3	S-109	2	1
60	CUT-OFF BAND SAW	3	MODEL SHOP	2	1
61	WYSONG POWER SHEAR	3	MODEL SHOP	2	1
62	DO ALL BAND SAW	3	MODEL SHOP	2	1

Figure 3

WORKLOAD ANALYSIS WORKSHEET DATE RUN : 03/21/84

Group No	Group Description	Total No devices due	overdue	PM std Hrs	**********Total Hours********** repair	wrkordr	regular	o'time	net
2	INDUSTRIAL MAINTNCE								
	Priority A	142	0	32.25					
	Priority B	61	0	28.20					
	Priority C	81	0	14.95					
	Priority D	10	0	5.60					
	Priority W	2	0	.80					
	Other	1	0	.40					
	Totals	297	0	82.20	0.00	0.00	0.00	0.00	-82.20

Figure 4

PREVENTATIVE MAINTENANCE COST & LABOR REPORT DATE RUN : 03/21/84 PAGE 1

DEVICE #	DESCRIPTION	ON SYS DATE	<----- PM YTD -----> LABOR HRS	MATERIAL $	<---- PM PRIOR ----> LABOR HRS	MATERIAL $	<----- REPAIR -----> LABOR HRS	MATERIAL $
1	HAND TAPER	883	.40	0.00	1.40	0.00	0.00	0.00
2	TOUCH TAPER 1	383	.40	0.00	1.90	0.00	0.00	0.00
3	TOUCH TAPER 2	383	.40	0.00	1.90	0.00	0.00	0.00
4	TOUCH TAPER 3	383	.40	0.00	2.25	0.00	0.00	0.00
5	TOUCH TAPER 4	983	.40	0.00	1.20	0.00	0.00	0.00
6	TOUCH TAPER 5	1283	.40	0.00	.20	0.00	0.00	0.00
7	TOUCH TAPER 7	284	0.00	0.00	0.00	0.00	0.00	0.00
14	DOMISTIC HOT WATER PUMP 1	882	.30	0.00	1.24	0.00	0.00	0.00
15	DOMESTIC HOT WATER PUMP 2	882	.30	0.00	1.24	0.00	0.00	0.00
16	DOMESTIC HOT WATER PUMP 3	882	.30	0.00	1.24	0.00	0.00	0.00
17	DOMESTIC HOT WATER PUMP 4	882	.30	0.00	1.24	0.00	0.00	0.00
18	DOMESTIC HOT WATER PUMP 5	882	.30	0.00	1.24	0.00	0.00	0.00
19	IRRIGATION PUMP	882	.30	0.00	1.28	0.00	.60	40.00
20	SUMP PUMP (PORTABLE)	883	.30	0.00	.40	0.00	0.00	0.00
39	AIR RECIVER TANK	384	0.00	0.00	0.00	0.00	0.00	0.00
40	COMPACTOR	882	1.60	0.00	6.80	0.00	0.00	0.00
41	TORK LIFT	882	1.60	0.00	4.25	0.00	0.00	0.00
42	HANKINSON AIR DRYER	882	1.25	0.00	2.35	0.00	0.00	0.00
43	ATLASCOPCO COMPRESSOR 2	882	1.25	0.00	8.40	91.50	0.00	0.00
44	INGERSOLL/RAND COMPRESSOR 1	882	1.25	0.00	3.75	0.00	0.00	0.00
45	HOUSE VACUUM	882	1.25	0.00	2.35	0.00	0.00	0.00
46	VACUUM PUMP 1	882	1.25	0.00	7.10	0.00	0.00	0.00
47	LIQUID RING VACUUM PUMP 2	882	1.25	0.00	4.40	3.00	1.30	63.00
48	EMERGENCY GENERATOR	882	1.20	0.00	6.95	0.00	0.00	0.00
60	CUT-OFF BAND SAW	882	.50	0.00	2.17	0.00	0.00	0.00
61	WYSONG POWER SHEAR	882	.50	0.00	2.47	0.00	0.00	0.00
62	DO ALL BAND SAW	882	.50	0.00	2.22	0.00	0.00	0.00

Figure 5

Testing compares new half-inch tape drive to existing drives

Bob Gilbert

Hewlett-Packard
Lake Stevens Instrument Division
Marysville, Washington
USA

Performance data on new products is an essential part of the development cycle. Hewlett-Packard recently introduced the HP 7978A, a one-half inch tape drive. Offered at half the price of the HP 7976A, the current high performance drive, the HP 7978A maintains approximately the same performance.

In order to validate the performance of the new tape drive, PCP (Plug Compatible Peripheral) tests were performed using six different filesets and eight different tape drives. The overall objective was to verify the performance of the HP 7978A, by determining sensitivity to fileset characteristics and system configuration and by making these measurements within a framework that allows comparisons with existing supported HP 3000 tape drives. Tests done in compliance with guidelines developed at HP's Computer Systems Division, where the HP 3000 is manufactured, measure performance in best-case and worst-case configurations (separate GIC versus shared GIC), using tape drives with the HP 3000/44.

SUMMARY OF RESULTS

The HP 7978 transfer data rate is within plus or minus 10 percent of the HP 7976 for the same filesets in a typical standalone STORE configuration. For a typical fileset as measured by the benchmark fileset #1, this data rate is 20.2 Mb/min (337 Kb/sec), as compared to 20.9 Mb/min (348 Kb/sec) for an HP 7976. Measurements were made on an HP 3000/44, running MPE V/P+ with an HP 7933 disc on a separate GIC from the HP 7978 tape drive. Measurements made earlier on an HP 3000/64 with an HP 7933 disc were only about two to three percent faster.

Configurations that are not optimized for backup performance will yield a lower performance than the above configuration (best case). One such configuration is specified in PCP performance testing, where the HP 7978 and disc(s) share a common GIC. In this case the HP 7978 has markedly lower data rates, as does the HP 7976.

This amounts to a worst-case configuration, where the system is incapable of supplying data fast enough for any GCR drive, regardless of streaming or nonstreaming mechanism. To accurately describe performance in this shared GIC configuration (or other nonoptimized configurations), a relative comparison of various tape drives is imperative.

DISCUSSION OF THE PERF60 GRAPH

Graph PERF60 will first be discussed before interpreting the results because it is a single, comprehensive (detailed) presentation of completed PCP testing. The graph's detail allows comparisons among the various HP tape drives currently supported by the HP 3000. The following text should make test result imterpretation clear and allow simple interpretation of other configurations/test. Definitions for notations included on the graph are:

- *BURST = tapespeed * density*
  The burst transfer rate is widely used in data sheets because it is the largest, most impressive performance number and does not reflect system-related performance dependencies. It is the maximum instantaneous writing date attainable within a record;
- *MAX = tapespeed*density**(record-length/(record-length+gap-length))
  This data rate is calculated for each drive according to its corresponding tape speed, maximum record size, and gap length. It is the maximum writing rate attainable within a file.

Six different filesets on eight different tape drives were run, using two different configurations. The two configurations, based on doing separate and shared GIC tests, were measured using an HP 3000/44 with an HP 7933 disc. Previous tests showed little performance difference between using an HP 3000/44 or HP 3000/64 computer; limitations appear to be I/O-related, and both computers share the same I/O architecture.

Filesets are annotated on the graph and are basically of two types; #1 is a typical customer fileset, and #2 through #5 are filesets with progressively smaller file sizes. Another fileset was involved in the testing, but since it ran at almost the same speed as #3, it was not graphed. Fileset #1 provides benchmarking of actual customer filesets and realistic estimation/comparison among HP's drives. Filesets #2 through #5 allow estimating sensitivity of (1) the system's ability to deal with progressively smaller files and (2) evaluation of how drives compare relative to each other within a given system configuration.

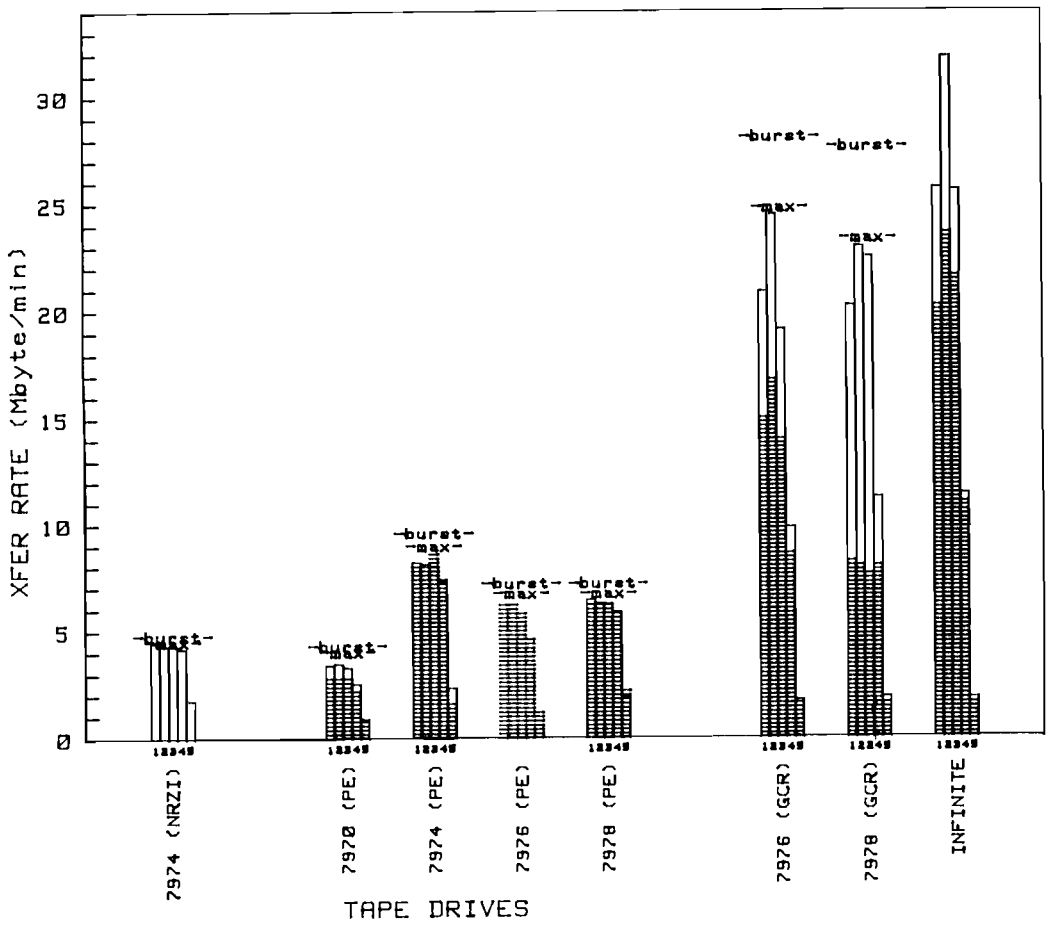
Tape drives are annotated along the x-axis, with designations of whether they operate in NRZI (800cpi), PE

(1600cpi), or GCR (6250cpi). It is interesting to note that GCR and PE formats on the HP 3000 allow 16,384 bytes and 8,192 bytes respectively as the maximum record sizes, while the HP 7974 and HP 7978 tape drives will support 16,384-byte records in either format. Therefore, about five percent more performance is available for PE formats.

The drive marked "INFINITE" needs more explanation. This is an HP 7978 prototype-running firmware with the write-record and write-file-mark procedures stubbed, so that after data has passed from the host to the drive it is not actually written on tape. This allows analysis of actual peak data-transfer rates given the configuration, fileset, tape drive, current STORE program architecture, and I/O channel protocol. It is a good comparison of how well a drive is performing relative to the HP 3000 system limit.

Each drive is graphed with the five filesets plotted in Mb/min with the HP 7933 and tape drive on separate GICs. Shaded portions within these measurements show the values for running the same stream job in a shared GIC configuration, with the HP 7933 and tape drive operating on the same GIC. Basic run information is annotated on the graph. Additional run related details follow the graph.

STORE performance measurements using HP 7970, 7974, 7976, and 7978 tape drives



Basic Run Information

1. BURST - density * tapespeed (calculated)
2. MAX - density * tapespeed * (1 - overhead)
   - calculated quantity
   - PE assumes 8192 bytes record and 0.6 inch gap
   - GCR assumes 16384 bytes record and 0.48 inch gap
3. Filesets
   - #1 @.@.FILES; 252 files @ 44.85 Mbyte
   - #2 @.@.PCPA0000; 21 files @ 52.43 Mbyte
   - #3 @.@.PCPB0000; 205 files @ 52.43 Mbyte
   - #4 @.@.PCPD0000; 2048 files @ 52.43 Mbyte
   - #5 @.@.PCPF0000; 20,480 files @ 52.43 Mbyte
4. INFINITE speeds are measured on a modified 7978
5. Measured single and multiple GIC configurations shaded portion shared GIC configurations
6. Mbyte = 1,000,000 bytes

--- KEY POINTS ---

1. The 7978 performs within +/-10% of 7976 for same filesets (non-shared GIC).
2. Performance degradation occurred for GCR drives in shared GIC configuration.

STORE performance measurements using HP 7970, 7974, 7976, and 7978 tape drives

1. HP 3000 information:
   - HP 3000/44 running MPE5/P+
   - STORE version = PO6PMPE4 (pass 2); SHOW option off
   - STORE fileset;*T;NUMBUF=X;TIME'
   - STORE listing provided the timings needed
   - STORE buffers = 6 @ 16kW (16kW= 32768bytes)
2. Disc information:
   - disc 1= 7925; system disc
   - disc 2= 7933; all test files loaded only on 7933
   - discs 1,2 share GIC 11
   - system reloaded

3. Tape drive information:
   - 7970; gap= .60 in; speed= 45 ips
   - 7974; gap= .60 in; speed= 50 ips stop/start; 100 ips streaming
   - 7976; PE gap= .60 in; GCR gap= .35 in; speed= 75 ips
   - 7978 (#0064); code=A0.00(final); PE gap= .60 in; GCR gap= .48 in; speed= 75 ips
   - INFINITE (modified 7978) write code inactive; protocol with host active
   - drives are on GIC 9 normally; on GIC 11 for shared tests
   - PE measurements used 8192 bytes/record; GCR used 16384 bytes/record
4. Workload estimate:
   - standalone environment; OP.BUCKIE and MANAGER.SYS only sessions active
5. Miscellaneous:
   - Dslines shut; disc caching off
   - Mb= 1,000,000 bytes
   - MAX value for each drive= 16kb records, no tape marks
   - drives in PE showed no performance gains for increasing the number of STORE buffers beyond the current value
   - multiple and single GIC configurations are plotted together.

SUMMARY OF PCP TEST RESULTS

The objective of PCP testing is to determine sensitivities of a peripheral tape drive to fileset characteristics and system configuration, and to make these measurements within a framework that will also allow comparisons with existing supported HP 3000 tape drives. Testing was initiated on earlier MPE-4 versions but completed on the latest release of MPE V/P+. Although two different MITs were used, it is interesting that speeds have varied only plus or minus three percent using identical test setups.

An "infinite speed" drive was created by modifying an HP 7978, thus allowing measurement of the STORE program architecture, tape driver, immediate response channel protocol, and HP 7978 firmware overhead. This allowed measurement of the maximum possible backup rates for a given fileset, workload, and host configuration/architecture. The HP 7978 runs within about 30 percent of the infinite data rate and the HP 7976 within 25 percent. The implication is that changes in the either the current STORE program or HP 7978 tape drive will result in small performance increases.

Fileset characteristics influence performance. Measurements were made on an HP 3000/44 with an HP 7933 disc (measurements made earlier on an HP 3000/64 with an HP 7933 disc were only about two to three percent faster). These tests were run using fileset #1, representing typical customer filesets, with all data contained on a single HP 7933 disc. The fileset was reloaded so locality was very high. Using this fileset and nine others, there was about a seven percent performance increase for the HP 7976 and about twenty percent increase for the HP 7978 for a reloaded versus highly fragmented system.

The HP 7978-GCR drive data rate is within plus or minus 10 percent of the HP 7976, for the same filesets in a typical standalone STORE configuration. This data rate is 20.2 Mb/min as compared to 20.9 Mb/min for an HP 7976.

PE drives are almost insensitive to fileset structure, due to the low data rates of the drives. See the following sections for more details.

EFFECT OF FILESET SIZE

To measure the effect of fileset size, several more filesets were created. Varying the size of files and keeping them at one extent/file allowed interpretation of the file boundary overhead between system and drive. For each fileset, look at the INFINITE drive to determine the maximum data rate the HP 3000 will deliver.

Fileset #2 is best case as it keeps doing contiguous disc I/Os without any file boundary overhead. In comparing the HP 7978-GCR and HP 7976-GCR, one sees almost no performance difference, and each drive operates within one percent of the theoretical maximum.

Fileset #1 (typical) or Fileset #3 (256,000 bytes/file) are not appreciably different.

Fileset #4, which is a 25,600-byte file, shows the effect of the file size on the system, because the INFINITE drive shows no higher data rate than the HP 7978-GCR.

Fileset #5, which is a 2,560-byte file, is totally system-limited, even for PE drives. PE drives perform at near their BURST data rates for all filesets except #5.

The transfer rate is so low for single-user workloads, as specified in PCP testing, that PE drives are relatively configuration insensitive. The HP 7974 streamed almost the entire time, even in shared GIC configurations.

STORE PERFORMANCE WEAKENED WITH SHARED GIC

Shared GIC configurations were measured not only for the typical fileset (Fileset #1) as specified by the PCP document, but for all filesets and for all of the drives. Many installations operate with the discs spread across all GICs, a configuration HP recommends for online disc I/O reasons, with the tape drive hooked up on the non-sysdisc GIC. In particular, beause the HP 3000/44 I/O system only allows two high speed GICs (Series 64 supports two IMBs and therefore four GICs), these installations in the worst case will see GCR backup performance not much different than PE performance.

So what does shared GIC testing mean? It is actually a lower bound for performance. If a site is set up with two GICs, an HP 7933 disc on each GIC, and a tape drive on one of these GICs, the system data rate will be higher than the above shared GIC PCP test results, but less than the multiple GIC configuration test results.

Shared GIC configurations for the HP 7978 should be avoided for performance-oriented systems, if possible. Even in this configuration, though, the HP 7978 for all filesets is faster than the HP 7974 (which is streaming). For multi-user or non-STORE tape operations, user will not see a great deal of performance difference between shared GIC or multiple GIC configurations, because data rates may be less than 160,000 bytes/sec.

CONCLUSIONS

Performance tests indicate that the HP 7978 data-transfer rate for the same filesets is typically within plus or minus 10 percent of the HP 7976, which validates the price/performance claims for the new drive.

Configuration and fileset sizes do affect performance. There is a performance degradation when the tape drive operates in a shared GIC configuration as opposed to being on a separate GIC. If STORE performance must be optimized, it is recommended that the tape drive operate on a separate GIC. For other operations, performance is not noticably different between shared and nonshared GIC configurations. It must be noted that this applies mainly to GCR operations; PE performance is relatively configuration-insensitive.

The structure of files within a fileset does directly affect the backup performance. As files become progressively smaller, performance worsens on all drives, dramatically so in the most extreme case of Fileset #5. Additionally, GCR drive performance increased when backup was done on a reloaded disc with high locality in comparison to a highly fragmented one. Because of the slower data rates of the PE drives, they are not really affected by fileset structure.

Pascal strings and MPE

Joseph Berry

Motorola/Hewlett-Packard
Tel Aviv, Israel

A number of examples are found in the Pascal 3000 manual that explain how to interface Pascal to the rest of the MPE world. Specifically, the appendices contain examples of interfacing to MPE intrinsics, IMAGE, V/PLUS, and so forth. In these examples, the variables used in the procedure calls are typically of type PACKED ARRAY of CHAR.

If one were to enter the IMAGE example that appears in the manual, a number of identical Pascal warnings would appear, saying:

BYTE TO WORD ADDRESS CONVERSION HERE (531)

The program works. Everything is as it shoud be. The listing, however, looks horrible, and there's a sense of untidiness in a computer listing with dozens of warnings. One also has to look through the listing carefully for any error that is not a #531.

After having encountered this problem too many times, I set out to find a solution. I've found two workable solutions thus far.

USING RECORD STRUCTURES

The following simple example demonstrates one solution. It is representative of how Pascal and IMAGE are used together:

```
$uslinit$
$standard_level 'HP3000'$
  program PASTEST(output);

  type
    pp = packed array [1..80] of char;
    single_int = -32768..32767;

  var
    base     : pp;
    password : pp;
    mode     : single_int;
    status   : packed array [1..10] of single_int;
    procedure dbopen;intrinsic;

  begin
    base     := ' BASE;';
    password := ';';
    mode     := 1;
    dbopen(base, password, mode, status);
    end.
```

When this program is compiled, it will generate the warning message described above. However, by using RECORD structures, we can force Pascal to recognize that we are indeed starting our variables on word boundaries. Here is an example of how the new program might appear.

```
$uslinit$
$standard_level 'HP3000'$
  program PASTEST (output);

  type
    pp = record
         db : packed array [1..80] of char;
         end;

    single_int = -32768..32767;

  var
    base     : pp;
    password : pp;
    mode     : single_int;
    status   : packed array [1..10] of single_int;

  procedure dbopen;intrinsic;

  begin
    base.db     := ' BASE;';
    password.db := ';';
    mode        := 1;
    dbopen(base, password, mode, status);
    end.
```

When compiled, this program produces no error messages. Everything works as advertised.

USING STRING DATA TYPE

The second technique is to take advantage of the Pascal's STRING data type. This solution is somewhat cleaner, because we do not have to create unnecessary RECORD types. The following example demonstrates this:

```
$uslinit$
$standard_level 'HP3000'$
  program PASTEST (output);

  type
    pp = string [80];
    single_int = -32768..32767;

  var
    base     : pp:
    password : pp;
    mode     : single_int;
    status   : packed array [1..10] of single_int;

  procedure dbopen;intrinsic;

  begin
    base     := ' BASE;';
    password := ';';
    mode     := 1;
    dbopen(base, password, mode, status);
    end.
```

Knowing that the above example compiles is only half the question. Will it execute properly? We are passing a data type (STRING) known only to Pascal to a non-Pascal procedure. In order to more fully understand the answer to the execution question, I conducted four tests. They demonstrate the usability of STRING data-types with non-Pascal procedures (specifically, with intrinsics).

When using arrays together with intrinsics (this includes IMAGE procedures), four different combinations exist. An array can either be a byte array or a logical array; it can be either an array that the intrinsic reads from or an array that the intrinsic writes to. The following examples demonstrate each case.

Passing a logical array to an intrinsic

The intrinsic PRINT reads data from a logical array and prints the contents to the CRT. The Pascal example of this case is presented as follows:

```
  var
    S1 : string[80];
  begin
    S1 := 'This line is to be printed.';
    print(S1, strlen(S1), 0);
  end.
```

The second parameter of the PRINT intrinsic is the length, in bytes, of the logical array. The function STRLEN satisfies this requirement. The code generated by Pascal is correct. Here is a sample of the code that is generated:

```
  LRA  DB+10,I {load the address of S1; strings are stored
                as with word addresses, not byte ad-
                dresses}
  INCA         {add one to the address; i.e., point to the
                data and not to the word containing the
                string length}
  LOAD DB+10,I{load the address of S1; this, of course,
                points to the length of the string; it's the
                second parameter in the PRINT intrin-
                sic}
  ZERO         {load zero on top of stack; this is the
                third parameter}
  PCAL PRINT
```

Passing a byte array to an intrinsic

The intrinsic BINARY converts ASCII digits into a pure binary number. BINARY expects the ASCII characters to be passed as a byte array. A Pascal example of this case is as follows:

```
  type
    single_int = -32768..32767;
  var
    S1 : string[80];
    N1 : single_int;
  begin
    S1 := '15';
    N1 := binary(S1, strlen(S1));
  end.
```

The second parameter of the BINARY intrinsic is the length, in bytes, of the logical array. The function STRLEN satisfies this requirement. The code generated by Pascal is again correct. Here is a sample of the generated code:

```
  LRA DB+10,I  {load the address of S1; strings are stored
                with word addresses, not byte addresses}
  LSL 1        {left-shift the address by one; this is equiv-
                alent to multiplying by two or converting
                a word address to a byte address}
  LDI 2        {place 2 on top of stack; see next line}
  LADD         {add 2 to the byte address; in other words,
                calculate the address of the data and not
                the address of the string length}
  LOAD DB+10,I {load the address of S1; now we want the
                address of the string length}
  PCAL BINARY
```

Reading a byte array from an intrinsic

In the previous two examples, we stored data into arrays that were then accessed by intrinsics. What about retrieving information from byte arrays that were filled by intrinsics? The ASCII intrinsic returns the character representation of a binary number in a byte array. The Pascal example is presented here as follows:

```
  type
    single_int = -32768..32767;
  var
    N1, N2 : single_int;
    S1     : string[80];
```

```
begin
  N2 := 23;
  N1 := ASCII(N2, 10, S1);
  setstrlen(S1, N1);
end;
```

Note that we have one peculiarity in this example. Specifically, we have to use the SETSTRLEN function after the ASCII intrinsic. The ASCII intrinsic returns the length of the string that was created from the binary number *N2*. Pascal has no way of knowing how long *S1* is or is supposed to be. Without initializing the length of the string *S1*, Pascal will use whatever value was in the length part of the variable. Typically, this will be either garbage or zero. The code generated by Pascal appears below.

```
LOAD DB+7   {load the address for N2}
LDI 12      {load the number 10, decimal, to top of
             stack}
LRA DB+10,I {load address of S1 onto the stack}
LSL 1       {left-shift one bit; i.e., multiply by two to
             get a byte address}
LDI 2       {put 2 onto the stack for later addition to
             point to data, not count}
LADD        {add 2 to the address of S1; i.e., point to
             data}
PCAL ASCII
```

Reading a logical array from an intrinsic

The problem that occurs in the third example above also occurs when reading data from a logical array. Specifically, the length previously established for the string array must be manually set. This is seen in the following example:

```
type
  single_int = -32768..32767;
var
  N1 : single_int;
  S1 : string[80];
begin
  N1 := read(S1, -80);
  setstrlen(S1, N1);
end.
```

In this example, we are reading up to 80 characters from $STDIN. The actual number of characters read are returned from the READ intrinsic in variable *N1*. The expanded assembly code is as follows:

```
LRA DB+10,I {load the top of stack with the address of
             S1}
INCA        {increment by one word to point to the
             data}
LDNI 120    {put -80 on top of stack}
PCAL READ
```

SUMMARY

We see from the above tests that working with STRING data types is an acceptable technique for communicating to the MPE world. It has its place not only in bypassing useless warning messages, but also in expediting programming.

The only caveat is that Pascal must be aware of the exact data types of the procedures being called. With intrinsics, this is straightforward since Pascal looks in the file SPLINTR.PUB.SYS for the intrinsic definitions. Users must be careful about defining their own non-Pascal procedures so that Pascal will generate the correct code.

JOURNAL

Introduction to systems programming on the HP 3000 using Pascal

Stephen Tucker

Faculty of Business
Royal Melbourne Institute of Technology
Melbourne, Australia

ABSTRACT

This paper introduces Pascal as a tool for systems programming on the HP 3000. It shows that Pascal contains features that are necessary and desirable for solving systems programming tasks.

INTRODUCTION: SOME GOOD REASONS FOR USING PASCAL

Pascal has not had a great penetration into the field of systems programming. This is attributable mainly to the view that strong data type rules and systems programming do not mix. This paper contends that strong typing and the many other features that exist in Pascal are not only desirable, they are essential where the programming of tasks is critical to the smooth and uninterrupted running of a computer.

Pascal guards the programmer from making mistakes, such as examining the wrong bit in a word, by making it easy for the programmer to define types that reflect the desired meaning of data stored in the computer. This means that the logic of interpreting information is embedded in the definition of a structure, rather than in the executable procedural portion of code. This also makes the meaning of a program much clearer to anyone reading the program.

PASCAL VARIABLE MAPPINGS

To successfully program systems applications in Pascal, it is essential to know what format the compiler will use for variable storage, and what addressing mode it will use to reference variables. The Pascal/3000 *Reference Manual* contains useful information on both these topics. The Pascal/3000 compiler option "$TABLES ON$" can also be used for verifying that the compiler has allocated storage in the manner intended for each particular variable.[1]

The following table summarizes Pascal type-storage allocations as given in the reference manual.[2]

	Storage Unit		
Type	**independent**	**unPACKED**	**PACKED**
boolean	1 word	1 byte	1 bit
integer	2 words	2 words	2 words
integer subrange	1/2 words	1 2 words	minimum bits
enumerated	1 word	1 byte 1 word	minimum bits
real	2 words	2 words	2 words
longreal	4 words	4 words	4 words
char	1 word	1 byte	1 byte
pointer	1 word	1 word	1 word

In most applications PACKED structures will be used. These are usually suitable, but have one major drawback: The packing of PACKED structures into PACKED structures is not performed. This means that some structures—such as the user attribute bits of the user capability word, which logically are one unit but do not occupy a full word—cannot be defined as a RECORD to stand alone. When a record such as this is included as an element in a more complete description, the Pascal/3000 compiler will align the sub element on a word boundary. Thus the correct mapping cannot be achieved.

Addressing mode considerations only becomes a factor when accessing structures outside the program's normal scope. In these cases, such as accessing system tables, it is necessary to ensure that the compiler will address the desired structure correctly.

By using pointer variables for system table access the compiler will generate the correct code, provided the correct DB relative address is placed into the pointer variable first. This is so because for all dynamic variables, that is,

pointer variables, the compiler must use indirect addressing.[3]

SOME EXAMPLES OF VARIABLE MAPPINGS

(1) *PACKED RECORDs/ARRAYs* of booleans for bit-maps, that is, the capability word passed from the WHO intrinsic.[4]

The following variable definition

```
user_capability :
  PACKED RECORD
    sm, am, al, gl, di, op, cv, uv, lg,
    res1, res2, res3, res4, cs, nd, sf,
    res5, res6, res7, res8, res9, res10, res11,
    ba, ia, pm, res12, res13, mr, res14, ds, ph : boolean;
  END;
```

will allow expressions of the following form

```
IF user_capability.am THEN...
```

(2) *Integer subranges* for small numbers contained in table definitions, that is, the system date passed from the CALENDAR intrinsic.[5]

The following variable definition

```
PACKED RECORD
  year_of_century : 0..99;
  day_of_year : 1..366;
END;
```

will map exactly onto the return from the CALENDAR intrinsic

(3) *Enumerated types* can be used when a clearer definition of the possible values of variables is desired, that is, the foptions parameter of the FOPEN intrinsic call can be defined as a PACKED RECORD of enumerated types.[6]

The following fragment of a variable definition

```
foptions :
  PACKED RECORD
    .
    .
    domain :
      (new_file,
     old_permenant_file,
     old_temporary_file,
     old_file);
  END;
```

will allow statements of the form

```
foption.domain:=new_file;
```

VARIABLE REDEFINITIONS

All the examples discussed so far use strict data structures that do not allow for different interpretations of the data represented. Yet in systems programming it is often necessary to interpret variables in different ways.

The mechanism in Pascal for achieving this is a RECORD structure using an undiscriminated union or, as the Pascal/3000 reference manual calls them, free union variant records.[7] Pascal's undiscriminated union allows the same variable to have different interpretations defined for it. The procedural code can then select any defined interpretation, by using the field identifier it requires.

A Simple Example

Consider the situation where you would like to see the binary representation of a decimal integer number. There are two basic ways to tackle this problem: The first uses shift (DIV / MOD) operations on the integer to isolate each binary digit; the other interprets the storage of that number in a different way.

Example 1 (see page 18) illustrates such a problem: All the information is really contained in the variable, and it is up to the programmer to determine how to interpret the information—decimal, binary, octal, or other formats. The idea of interpreting variables in several ways, using an undiscriminated union, is central to systems programming in Pascal. With this method, it is possible to do machine level referencing, while still retaining the strong typing, range checking qualities of Pascal.

A More Advanced Example

The whole concept of several different interpretations for a single variable is important to systems programming, because, at the machine level, this is what actually occurs. Consider a word in memory: The value stored in that word could have many different meanings depending on the interpretation. It could contain a machine instruction, ASCII characters, a numeric value, and so forth, and these could then be subdivided again. For instance, a numeric value could be data for a program or it could be the address of data for a program.

Example 2 (see page 20) illustrates one such situation. This example accesses MPE's PXGLOB table, which exists for every user process in the area below DL on the user's stack.[8] The PXGLOB table is part of the PCBX table, and its address can be found in the PCBX table at memory location DL-1. The address found at this location is a DL relative address, and so must be subtracted from the value of the DL register, to obtain a correct DB relative address. Studying Example 2, we can see that the table accessing variable is used in three different ways:

- as an integer subrange so that arithmetic can be performed using the variable as the result;
- as a pointer to an integer subrange so that indexed address references can be performed;

- as a pointer to the PXGLOB table, the real purpose of the exercise.

It should be noted that all three representations occupy exactly one word of memory on the HP 3000.

Some other points to note about Example 2 are that the value of the DL register is obtained as the result of an SPL function. Pascal can't perform some operations properly, such as obtaining the values of registers. The other point to note is that the procedure needs to be in privilege mode to address beyond the DL register.

SUMMING UP

The examples described demonstrate only a tiny range of systems programming applications using Pascal. The business faculty at Royal Melbourne Institute of Technology is involved in building a library of Pascal types and high-level procedures suitable for systems programming and interfacing with MPE. The systems programming types can be included in programs using Pascal/3000's "include" compiler command, while the systems programming procedures are stored in RLs, ready to be included at "PREP" time.

The key quality these types and procedures are striving for is readability and maintainability. Pascal has helped us more than any other language available on the HP 3000 to achieve these goals.

EXAMPLE 1

HEWLETT-PACKARD, HP32106A.00.05, PASCAL/3000, © HEWLETT-PACKARD CO.

```
 1.000   0  0  $uslinit$
 2.000   0  0
 3.000   0  0  PROGRAM word_representation(input,output);
 4.000   0  0
 5.000   0  0  { This program is intended to demonstrate the possibilities of the Pascal undiscriminated union by
 6.000  **  0    presenting a simple example. The example takes a one word integer value and prints out its decimal
 7.000  **  0    and binary values without any conversion calculations taking place.
 8.000  **  0
 9.000  **  0
10.000  **  0
11.000  **  0
12.000  **  0
13.000  **  0
14.000  **  0
15.000  **  0
16.000   0  0  }
17.000   0  0
18.000   0  0  CONST
19.000   0  0    bits_per_word = 16; {number of bits to HP 3000 word}
20.000   0  0
21.000   0  0  TYPE
22.000   0  0    binary_digit = 0..1;
23.000   0  0
24.000   0  0    decimal_number =   {one word integer representation}
25.000   0  0      -32768..32767;
26.000   0  0
27.000   0  0    binary_number = {one word binary representation}
28.000   0  0      PACKED ARRAY [1..bits_per_word] OF binary_digit;
29.000   0  0
30.000   0  0    word_representations = {possible word representations}
31.000   0  0      ( binary, decimal );
32.000   0  0
33.000   0  0    word =     {actual word representations}
34.000   0  0      RECORD
35.000   0  0        CASE word_representations OF
36.000   0  0          binary :
37.000   0  0            (binary_representation : binary_number);
```

```
 38.000  0  0            decimal :
 39.000  0  0              (decimal_representation : decimal_number);
 40.000  0  0        END;
 41.000  0  0
 42.000  0  0    VAR
 43.000  0  0      sample : word;
 44.000  0  0        index : integer;
 45.000  0  0
 46.000  0  1    BEGIN
 47.000  0  1      writeln(output,
 48.000  1  1        ' Sample Decimal and Binary Representations');
 49.000  1  1      writeln(output);
 50.000  2  1
 51.000  2  1      WHILE NOT eof(input) DO
 52.000  3  2        BEGIN
 53.000  3  2          readln(input, sample.decimal_representation);
 54.000  4  2          writeln(output);
 55.000  5  2          writeln(output, ' Decimal value is ',
 56.000  6  2            sample.decimal_representation:16);
 57.000  6  2          write(output, ' Binary value is ');
 58.000  7  2          FOR index:=1 TO bits_per_word DO
 59.000  8  2            write(output,
 60.000  9  2              sample.binary_representation[index]:1);
 61.000  9  2          writeln(output);
 62.000  9  2          END;
 63.000  9  1    END.
              NUMBER OF ERRORS = 0          NUMBER OF WARNINGS = 0
              PROCESSOR TIME 0: 0: 3        ELAPSED TIME 0: 0: 7
              NUMBER OF LINES = 63          LINES/MINUTE = 1260.0
```

Run of the above program

```
Sample Decimal and Binary Representations

Decimal value is                    0
Binary value is      0000000000000000

Decimal value is                    2
Binary value is      0000000000000010

Decimal value is                    4
Binary value is      0000000000000100

Decimal value is                    8
Binary value is      0000000000001000

Decimal value is                   16
Binary value is      0000000000010000

Decimal value is                32767
Binary value is      0111111111111111

Decimal value is               -32768
Binary value is      1000000000000000
```

Example by:
Stephen Tucker
Faculty of Business,
R.M.I.T.

EXAMPLE 2

```
PROCEDURE read_pxglob(VAR pxglob_table : pxglob);

  { This procedure 'reads' the pxglob table, it returns the table value in the variable pxglob_table
  }
  TYPE
      smallint = -32768..32767

      ptr_types = ( table_ptr, indexed_address, value );

      pointer =
        RECORD
          CASE ptr_types OF
            table_ptr        : (pxglob_ptr: ^pxglob);
            indexed_address : (address : ^smallint);
            value            : (ptr_value : smallint);
        END;

  VAR
      pxglob_pointer : pointer;

  FUNCTION dlregister : smallint;
              external spl;

  BEGIN
      pxglob_pointer.ptr_value := dlregister - 1;
      pxglob_pointer.ptr_value := dlregister - pxglob_pointer.address^;
      pxglob_table := pxglob_pointer.pxglob_ptr^;
  END;
```

NOTES

1. *HP 3000 Computer Systems: Pascal/3000 Reference Manual* (Cupertino, Calif.: Hewlett-Packard, 1981), p. 8-47.
2. Ibid., pp. 9-2–9-10.
3. Ibid., p. 9-23.
4. *HP 3000 Computer Systems: Intrinsics Reference Manual,* 3rd ed., update #1 (Cupertino, Calif.: Hewlett-Packard, 1981), p. 2-195.
5. Ibid., p. 2-15.
6. Ibid., p. 2-80.
7. *Pascal/3000 Reference Manual,* pp. 2-13, 2-31.
8. *Hp 3000 Computer Systems: Systems Tables Reference Manual,* 2nd ed. (Cupertino, Calif.: Hewlett-Packard, 1981), pp. 7-10.

REFERENCE

Jensen, K., and N. Wirth. *Pascal User Manual and Report.* New York: Springer Verlag, 1975.

INTEREX BOARD OF DIRECTORS

Journal
Interex
2570 El Camino Real West, Fourth Floor
Mountain View, California 94040
U.S.A.