Worldwide Response Center

HP 3000 APPLICATION NOTE #85

The Optimization of Programs in MPE/XL

HEWLETT
PACKARD

March 15, 1991
Document P/N #5960-4335

RESPONSE CENTER APPLICATION NOTES

HP 3000 APPLICATION NOTES *are published by the Worldwide Response Center and are distributed with the Software Status Bulletin. These notes address topics where the volume of calls received at the Center indicates a need for addition to or consolidation of information available through HP support services.*

Following this publication you will find a list of previously published notes and a Reader Comment Sheet. You may use the Reader Comment Sheet to comment on the note, suggest improvements or future topics, or to order back issues. We encourage you to return this form; we'd like to hear from you.

The Optimization of Programs In MPE/XL

Introduction

To optimize or not to optimize? That is the question!

In MPE/XL, the native compilers (PASCAL/XL, COBOL/XL, FORTRAN/XL and C/XL) offer the possibility of generating optimized code. The intent of this Application Note is to present the functions of the optimizer and to study the different levels of optimization.

How to Use the Code Optimizer?

To invoke the code optimizer during compilation, a compilation option needs to be added to the beginning of the source program.

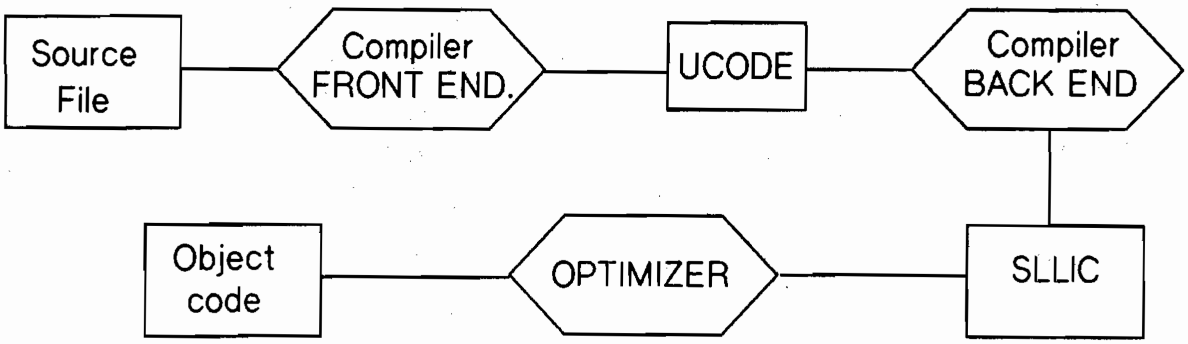
The syntax of compilation options by language is given in the figure below.

```
PASCAL/XL : $OPTIMIZE    ('LEVEL0')   no optimization
                         ('LEVEL1')  optimization level 1
                         ('LEVEL2')  optimization level 2
                         (  ON    )  optimization level 2
                         (  OFF   )  no optimization


  COBOL/XL   : $CONTROL (OPTIMIZE  )  optimization level 1
                        (OPTIMIZE=0)  no optimization
                        (OPTIMIZE=1)  optimization level 1

  FORTRAN/XL: $OPTIMIZE (  ON    )    optimization level 2
                        (  OFF   )    no optimization
                        (LEVEL1  )    optimization level 1
                        (LEVEL2  )    optimization level 2

  C/XL       : #pragma OPT_LEVEL (  1  ) optimization level 1
                                 (  2  ) optimization level 2
                                 (  ON ) optimization level 2
                                 ( OFF ) no optimization
```

When Does the Optimizer Intervene

Let us begin by detailing the phases of a compilation.

The diagram below shows the different phases which take place between the submission of a source file to the compiler, and the generation of a object code file (SOM: System Object Module).



The first step of this operation is the translation of the source file by the compiler FRONT END.

This operation is the only one dependant on the language used since the result (UCODE) should be the same whatever the language of origin.

This design choice implies that all the following operations (BACK END, OPTIMIZATION) will be identical whatever the language. The UCODE is an intermediate internal code generated in an unnamed file which is then submitted to the BACK END during the compilation time. When compiling a very large program on screen, you may see a longer or shorter delay between the display of the last line of the source and the end of the compilation, this time, in fact, is the run time of the BACK END.

The BACK END therefore interprets the UCODE and generates the SLLIC (Spectrum Low Level Intermediate Code). This code is actually made up from groups of instructions which are ready, either to be optimized if the optimization level is 1 or 2, or to be regrouped in OBJECT MODULE.

The remainder of this Application Note we shall concentrate on the optimization phase.

Study of the Different Levels of Optimization

LEVEL 0

No optimization or OPTIMIZATION 0 level (default)

This mode is used during development and program DEBUGGING. In this case, the optimizer still intervenes for 5 operations.

Construction of basic blocks:

This operation consists of creating the data structures which divide the program and cut it into sections known as Basic blocks.

Registers allocation:

Here the optimizer on one hand assigns the necessary registers for the calculations and expressions evaluation, and on the other hand generates the procedural headers and footers.

Branch reduction:

During this phase the optimizer analyses the branches generated by the FRONT END and checks if a LONG type branch can be changed for a SHORT type and conversely.

Branch simplification:

This operation consists of extracting, from the instruction blocks between two branches, all those which can be extracted from it.

This then permits the long type branches to be changed to the short type (see example 1).

```
example 1 :
           Before operation                         After operation

            begin                                     begin
            var1 := 1                                 var1 := 1
            input var2                                input var2
            if var2> 0 then                           var3 := var1 + var2
              begin                                   if var2 > 0 then
              var3 := var1 + var2                       begin
              var4 := var2 * 2                          var4 := var2 * 2
              var4 := var4 + var3                       var4 := var4 + var3
              end;                                      end;
              write var4;                               write var4;
            end.                                      end.
```

This example, to simplify reading, is in high level language instead of machine language. You will note that the line

```
var3 := var1 + var2
```

is extracted from the instruction block comprised between BEGIN and END since this expression does not depend on the result of the test and is not used afterwards.

Thus, if the test result is negative, the number of instructions of the IF loop to not execute is reduced. This may permit passing from a long type branch to a short type.

Extension of PSEUDO INSTRUCTIONS:

The optimizer generates machine code for multiplication and division. For example: multiplication by 2 becomes bit shifts, multiplications by 3 become multiplication by 2 plus an addition.

```
(3*1 <==>2* 1+1 etc...)
```

LEVEL 1

Peephole optimizer:

This operation consists of checking if in the code, the addressing method of an instruction can be improved, if a code sequence can be shortened thanks to the use of instructions accessing or manipulating bits group.

Optimizing of branches:

Please note that in example 2 the group of instructions containing instr2 is NEVER used, consequently, as well as the direct branching at the end of the test near label 20, this instruction block is purely and simply deleted.

example 2 :

```
             Before                             After

        if a > 0 goto 10                   if a > 0 goto 20
             .                                  .
           instr1                             instr1
             .                                  .
     10 goto 20                         20 a := 1
             .
           instr2
             .
     20 a := 1
```

Improvement of instruction sequences:

This operation is necessary to avoid the conflicts between registers, also known as the INTERLOCKS REGISTER. In fact, an instruction accessing a register (LOAD, STORE) is composed of two phases. One FETCH phase, which fetches the information to be LOADed or to be STOREd and an EXEC phase which carries out the operation.

The RISC architecture allows the execution of the EXEC phase of an instruction when the FETCH phase of the next instruction has already begun.

(This system also has the name of PIPELINE architecture on other systems).

This procedure is only possible if both instructions in sequence *do not* manipulate the same registers, if this is the case, we are confronted by a case of REGISTER INTERLOCK. The optimizer function, therefore, is to modify the order of the instructions to favor the PIPELINE function while avoiding the REGISTER INTERLOCKS.

example 3:

```
          Calculation of  D := A + B + C

          Before Optimization

     LOAD <A>,R19      ; Register R19 loaded with contents of A
     LOAD <B>,R20
     ADD R19,R20,R21   ; Register R21 loaded with sum of registers R19 et R20
                       ; here we see a register interlock on register
```

```
                      ; R20 :the 'FETCH' phase of ADD cannot begin as soon
                        as the 'EXEC' phase of the previous LOAD is not
                        finished
LOAD <C>,R22
ADD R21,R22,R23
STORE R23,<D>       ; D loaded with contents of register R23.

     After Optimization

LOAD <A>,R19
LOAD <B>,R20
LOAD <C>,R22
ADD R19,R20,R21 ; In this case the ADD 'FETCH' phase is executed in
                  parallel with the previous LOAD 'EXEC' phase, this
                ; is because there is no register interlock between
                ; these two instructions
ADD R21,R22,R23
STORE R23,<D>
```

You will note here that the number of executed instructions is the same before optimization as afterwards, however in the second version, the instruction flow is executed more quickly, using all the power of RISC architecture.

Deletion of NOP instructions (No OPeration):

The BACK END, for reasons of simplicity, generates a large number of NOP instructions which are deleted when possible.

Deletion of extraneous code:

Notice in one of the above examples, how a group of unused instructions may be deleted.

LEVEL 2

At this level of optimization, the compilation consumes a great deal more memory resources. This is because each procedure is processed by the compiler as an entire unit of code. This is why compilation at this level is much longer and its use is recommended only when your programs are completely DEBUGged and tested. Furthermore, optimized code is extremely difficult to read.

Optimization is always carried out from the SLLIC, and each procedure is first treated as a unit of code. These units are then regrouped, and if necessary some optimizations are still carried out in such a way as to generate the best possible code.

Two concepts are used by the optimizer to produce the code, and these two concepts generate an analysis for each unit of code.

The analysis of the program control flux:

The purpose of this analysis is to divide the units of code into groups of instructions with the following characteristics: if the first instruction is executed, no event can stop the last instruction being executed.

Analysis of the data control flow:

This consists for the optimizer in searching among the data used in a code unit which is the most often referenced. This process allows the definition of which data it is judicious to reserve registers for in relation to those that can be left in memory.

Once both these analyses have been carried out, the optimizations already discussed in level 0 and level 1 are carried out.

Then specific operations intervene at level 2. We will look at these in more detail.

Optimization of registers allocation and reduction of memory transfers:

We see in example 4 below, that the variables a, b, and c are internal to this procedure, consequently it is not necessary to store them in memory (even if the program's MAP indicates that they are supposed to be there). This avoids 3 STORE instructions which are extraneous in the sense that these variables do not have functions external to the procedure. Furthermore, only 3 registers are required instead of 5.

```
example 4:
          source:begin
                 a:= parm1
                 b:= parm2
                 c:= a + b
                 parm4:= parm3 + c
                 end.

     Generated code before optimization:     Generated code after optimization

               LOAD <parm1>, R4                    LOAD <parm1>,R4
               STORE R4,<a>                        LOAD <parm2>,R5
               LOAD <parm2>,R5                     LOAD <parm3>,R6
               STORE R5,<b>                        ADD R4,R5,R4
               ADD R4,R5,R6                        ADD R4,R6,R4
               STORE R6,<c>                        STORE R4,<parm4>
               LOAD R7,<parm3>
               ADD R6,R7,R8
               STORE R8,<parm4>
```

Pre-evaluation of constant expressions:

Here, instead of re-evaluating the expression (a+b), it is calculated by the optimizer and directly integrated into the code.

```
example 5:
                Before                           After

                a:=1                             a:=1
                b:=2                             b:=2
                c:=a+b+param                     c:=3+param
```

Elimination of common sub expressions:

To do this the optimizer will search the code for all identical groups of instructions and substitute calculated instructions. The interest of this is that all redundancy in the code is avoided.

```
example 6:
                        Before                                  After

                      a := a + (c*b)                          t := c*b
                      d := d - (c*b)                          a := a+t
                                                              d := d-t
```

Here the optimizer prefers to reserve a temporary register to store the result (c*b). This avoids carrying out the calculation twice.

Deleting of code not dependant on loop contents:

The expression of the calculation of a is extracted here from the loop and its evaluation carried out once instead of 100.000 times.

```
example 7:
          Before                                  After

              For i:= 1 to 100000 do                a:=b+c*(e+f/h)
                begin                               for i:= 1 to 100000 do
                a:=b+c*(e+f/h)                        begin
                i:=a+i*2+a*(i-1)                      i:=a+i*2+a*(i-1)
                end                                   end
```

Development of induced variables

For this the optimizer uses what it knows about the type and format of the variables. In example 8 below, the optimizer uses the fact that an integer is coded on four bytes.

```
example 8:
          Source :
               procedure test (B,C:packed array[1..10] of integer) ;
               type tab= packed array [1..10] of integer;
               var  A : tab;
                    i     : integer ;
               begin
               for i:= 1 to 10 do
                 A(i) := B(i) + C(i);
               end.

          Generated code without optimization .

               B1 : T := i * 4
                    A(T) := B(T) + C(T)
                    i := i + 1
                    if i <= 10 goto B1
```

```
Generated code using induced variables

          B1 : T := T + 4
               A(T) := B(T) + C(T)
               if T >= 36 goto B1

          (T is a byte offset to the beginning of the array.)
```

We see here that the variable i is deleted thanks to the fact that the optimizer knows the length of an element of the array is four bytes. Further, we note that the multiplication by four (2 bit shifts) is replaced by an addition which will be carried out by a single instruction.

Use of Different Levels of Optimization

The 0 level of optimization must be used during the development phase of a program. When the program has been entirely corrected and is ready for production, it then suffices to recompile it with the adequate option to optimize at the highest possible level in the language used (2 for PASCAL, FORTRAN or C, 1 for COBOL). If a malfunction occurs at this time, either from the compiler or the program, you should reduce the optimization level and re-test.

We have effectively seen above that the levels of optimization 1 and 2 work on a certain number of different types of improvements. These optimizations may be contradictory in certain cases this is why the optimizer must sometimes arbitrate between several different optimization concepts.

Thus, it can happen that the choice having been badly made by the optimizer it is then necessary to recompile at a lower level.

Optimization Example

Here and on the following pages you will see two examples of the compilation of the same program, one without optimization and the other with a level 2 optimization.

These examples show the difference in size of the generated code.

```
NON OPTIMIZED VERSION
          MON, DEC 18, 1989,  6:14 PM

    0     1.000  0   $optimize 'LEVEL0'$
    0     2.000  0   $list_code on$
    0     3.000  0   program prog;
    0     4.000  0   var i,j,k: integer;
    3     5.000  1   begin
    3     6.000  1   i:=1;
    4     7.000  1   j:=4;
    5     8.000  1   k:=i+j;
    6     9.000  1   for i:=1 to 1000 do
    7    10.000  2     begin
    7    11.000  2     j:=8;
    8    12.000  2     k:=i+j;
    9    13.000  2     k:=4*j - j+12*6;
   10    14.000  2     end;
   10    15.000  1   end.
```

```
   10     16.000    0
                              NUMBER OF ERRORS =  0     NUMBER OF WARNINGS =  0
                              PROCESSOR TIME 0: 0: 1    ELAPSED TIME 0: 0: 1
                              NUMBER OF LINES =   16    LINES/MINUTE =  1814.7
                              NUMBER OF NOTES =    0
END OF COMPILE
:link po,px
HP Link Editor/XL (HP30315A.01.08) Copyright Hewlett-Packard Co 1986

LinkEd> link po,px

:do run
:run px;debug

DEBUG/XL A.1A.16
DEBUG Intrinsic at: 399.00005040 ?PROGRAM
$1 ($49) nmdebug > s
$2 ($49) nmdebug > s
$3 ($49) nmdebug > dc PROGRAM aa
PROG $399.50f8
000050f8  PROGRAM  6bc23fd9  STW       2,-20(0,30)
000050fc  PROGRAM+$4  37de0060  LDO       48(30),30
00005100  PROGRAM+$8  6bc03ff9  STW       0,-4(0,30)
00005104  PROGRAM+$c  e85f1edd  BL        ?_start+$1c,2
00005108  PROGRAM+$10  08000240  OR        0,0,0
0000510c  PROGRAM+$14  e85f1f0d  BL        ?_start+$3c,2
00005110  PROGRAM+$18  08000240  OR        0,0,0
00005114  PROGRAM+$1c  34010002  LDO       1(0),1
00005118  PROGRAM+$20  6b610020  STW       1,16(0,27)
0000511c  PROGRAM+$24  341f0008  LDO       4(0),31
00005120  PROGRAM+$28  6b7f0018  STW       31,12(0,27)
00005124  PROGRAM+$2c  4b730020  LDW       16(0,27),19
00005128  PROGRAM+$30  4b740018  LDW       12(0,27),20
0000512c  PROGRAM+$34  0a930e15  ADDO      19,20,21
00005130  PROGRAM+$38  6b750010  STW       21,8(0,27)
00005134  PROGRAM+$3c  34160002  LDO       1(0),22
00005138  PROGRAM+$40  6b760020  STW       22,16(0,27)
0000513c  PROGRAM+$44  34010010  LDO       8(0),1
00005140  PROGRAM+$48  6b610018  STW       1,12(0,27)
00005144  PROGRAM+$4c  4b7f0020  LDW       16(0,27),31
00005148  PROGRAM+$50  4b730018  LDW       12(0,27),19
0000514c  PROGRAM+$54  0a7f0e14  ADDO      31,19,20
00005150  PROGRAM+$58  6b740010  STW       20,8(0,27)
00005154  PROGRAM+$5c  4b750018  LDW       12(0,27),21
00005158  PROGRAM+$60  08150e96  SH2ADDO   21,0,22
0000515c  PROGRAM+$64  4b610018  LDW       12(0,27),1
00005160  PROGRAM+$68  08360c1f  SUBO      22,1,31
00005164  PROGRAM+$6c  b7f30890  ADDIO     72,31,19
00005168  PROGRAM+$70  6b730010  STW       19,8(0,27)
0000516c  PROGRAM+$74  4b740020  LDW       16(0,27),20
```

```
00005170  PROGRAM+$78  341507d0  LDO       1000(0),21
00005174  PROGRAM+$7c  82b4201a  COMBT,=,N20,21,PROGRAM+$90
00005178  PROGRAM+$80  4b760020  LDW       16(0,27),22
0000517c  PROGRAM+$84  b6c10802  ADDIO     1,22,1
00005180  PROGRAM+$88  6b610020  STW       1,16(0,27)
00005184  PROGRAM+$8c  e81f1f67  B,N       PROGRAM+$44
00005188  PROGRAM+$90  e85f1e55  BL        ?_start+$5c,2
0000518c  PROGRAM+$94  08000240  OR        0,0,0
00005190  PROGRAM+$98  08000240  OR        0,0,0
00005194  PROGRAM+$9c  e85f1e7d  BL        ?_start+$7c,2
00005198  PROGRAM+$a0  08000240  OR        0,0,0
0000519c  PROGRAM+$a4  4bc23f79  LDW       -68(0,30),2
000051a0  PROGRAM+$a8  e840c000  BV        0(2)
000051a4  PROGRAM+$ac  37de3fa1  LDO       -48(30),30

$4 ($49) nmdebug > e

END OF PROGRAM

OPTIMIZED VERSION
:pasxl ps,po
PAGE   1  HP PASCAL/XL HP31502A.01.21    COPYRIGHT HEWLETT-PACKARD CO. 1986
          MON, DEC 18, 1989,  6:04 PM

    0     1.000   0   $optimize 'LEVEL2'$
    0     2.000   0   $list_code on$
    0     3.000   0   program prog;
    0     4.000   0   var i,j,k: integer;
    3     5.000   1   begin
    3     6.000   1   i:=1;
    4     7.000   1   j:=4;
    5     8.000   1   k:=i+j;
    6     9.000   1   for i:=1 to 1000 do
    7    10.000   2     begin
    7    11.000   2     j:=8;
    8    12.000   2     k:=i+j;
    9    13.000   2     k:=4*j - j+12*6;
   10    14.000   2     end;
   10    15.000   1   end.
                      NUMBER OF ERRORS =  0     NUMBER OF WARNINGS =  0
                      PROCESSOR TIME 0: 0: 1    ELAPSED TIME 0: 0: 3
                      NUMBER OF LINES =   16    LINES/MINUTE =  1511.8
                      NUMBER OF NOTES =    0

END OF COMPILE
:do lin
:link po,px
HP Link Editor/XL (HP30315A.01.08) Copyright Hewlett-Packard Co 1986

:do ru
:run px;debug
```

```
DEBUG/XL A.1A.16

DEBUG Intrinsic at: 439.00005040 ?PROGRAM
$1 ($2e) nmdebug > s
$2 ($2e) nmdebug > s
$3 ($2e) nmdebug > dc PROGRAM 20
PROG $439.50f8
000050f8  PROGRAM  6bc23fd9  STW      2,-20(0,30)
000050fc  PROGRAM+$4  37de0060  LDO      48(30),30
00005100  PROGRAM+$8  6bc03ff9  STW      0,-4(0,30)
00005104  PROGRAM+$c  e85f1edd  BL       ?_start+$1c,2
00005108  PROGRAM+$10  08000240  OR      0,0,0
0000510c  PROGRAM+$14  e85f1f0d  BL      ?_start+$3c,2
00005110  PROGRAM+$18  08000240  OR      0,0,0
00005114  PROGRAM+$1c  341f0002  LDO     1(0),31
00005118  PROGRAM+$20  341707d0  LDO     1000(0),23
0000511c  PROGRAM+$24  82ff2012  COMBT,=,N31,23,PROGRAM+$34
00005120  PROGRAM+$28  b7ff0802  ADDIO   1,31,31
00005128  PROGRAM+$30  b7ff0802  ADDIO   1,31,31
0000512c  PROGRAM+$34  e85f1f0d  BL      ?_start+$5c,2
00005130  PROGRAM+$38  08000240  OR      0,0,0
00005134  PROGRAM+$3c  e85f1f3d  BL      ?_start+$7c,2
00005138  PROGRAM+$40  08000240  OR      0,0,0
0000513c  PROGRAM+$44  4bc23f79  LDW     -68(0,30),2
00005140  PROGRAM+$48  e840c000  BV      0(2)
00005144  PROGRAM+$4c  37de3fa1  LDO     -48(30),30
$4 ($2e) nmdebug > e

END OF PROGRAM
```

You will notice that the optimized version requires only 20 instructions instead of 44 in the basic version.

Published Application Notes

HP 3000

Following is a list of the Application Notes published to date. If you would like to order single copies of back issues please use the *Request Form* attached and indicate the number(s) of the note(s) you need, and the part number(s).

Note #	Part Number	Topic
1	5958-5824	Printer Configuration Guide - Version 1
2	5960-2841	Terminal types for HP 3000 HPIB Computers - Version 1
3	5960-2842	Plotter Configuration Guide
4	5960-2843	Printer Configuration Guide - Version 2
5	5960-2844	MPE System Logfile Record Formats
6	5960-2845	Stack Operation
7	5960-2846	COBOL II/3000 Programs: Tracing Illegal Data
8	5960-2847	KSAM Topics: COBOL's Index I/O: File Data Integrity
9	5960-2848	Port Failures, Terminal Hangs, TERMDSM
10	5960-2849	Serial Printers - Configuration, Cabling, Muxes
11	5960-2850	System Configuration or System Table Related Errors
12	5960-2851	Pascal 3000 - Using Dynamic Variables
13	5960-2852	Terminal Types for HP 3000 HPIB Computers - Version 2
14	5960-2853	Laser Printers - A Software and Hardware Overview
15	5960-2854	FORTRAN Language Considerations - A Guide to Common Problems
16	5960-2855	IMAGE: Updating to TurboIMAGE & Improving Database Loads
17	5960-2856	Optimizing VPLUS Utilization
18	5960-2857	The Case of the Suspect Track for 792X Disc Drives
19	5960-2858	Stack Overflows: Causes & Cures for COBOL II Programs
20	5960-2859	Output Spooling
21	5960-2860	COBOLII and MPE Intrinsics
22	5960-2861	Asynchronous Modems

HP 3000 (continued)

Note #	Part Number	Topic
23	5960-2862	VFC Files
24	5960-2863	Private Volumes
25	5960-2864	TurboIMAGE: Transaction Logging
26	5960-2865	HP 2680A, 2688A Error Trailers
27	5960-2866	HP Trend: An Installation and Problem Solving Guide
28	5960-2867	The Startup State Configurator
29	5960-2868	A Programmer's Guide to VPLUS 3000
30	5960-2869	Disc Cache
31	5960-2870	Calling the CREATEPROCESS Intrinsic
32	5960-2871	Configuring Terminal Buffers
33	5960-2872	Printer Configuration Guide - Version 3
34A	5960-2873	RIN Management (Using COBOLII Examples) (A)
34B	5960-2874	Process Handling (Using COBOLII Examples) (B)
35	5960-2875	HPDESK IV (Script files, FSC, and Installation Considerations)
34C	5960-2876	Extra Data Segments (Using COBOLII Examples) (C)
36	5960-2877	Tips for the DESK IV Administrators
37	5960-2878	AUTOINST: Trouble-free Updates
38	5960-2879	Store/Restore Errors
39	5960-2880	MRJE Emulates a HASP Workstation
40	5960-2881	HP 250 / 260 to HP 3000 Communications Guidelines
41	5960-2882	MPE File Label Revealed
42	5960-2883	System Interrupts
43	5960-2884	Run Time Aborts
44	5960-2885	HPPA Patching Conventions for HP3000 900 Series Processors - Version 1
45	5960-2886	Vplus & Multiplexers
46	5960-2887	Setting Up an HPDesk HPTelex for the First Time
47	5960-2900	Customizing Database Data Items & Changing Passwords in JCL Files
48	5959-9215	Printer Configuration - Version 4
49	5959-9227	Configuring DATACOMM Products Into MPE
50	5959-9228	VFC's for Serial Printers

HP 3000 (continued)

Note #	Part Number	Topic
51	5959-9237	Terminal Types for the HP 3000 HPIB Computers
52	5959-9242	Configuring MRJE
53	5959-9245	Using Special Characters on the 700/9x Series Terminals
54	5959-9251	Improving Database Performance
55	5959-9258	Customized Message Catalogs and Help Facilities
56	5959-9266	BRW Tips for Beginners
57	5959-9270	Configuring the HP 2334A Plus & HP 2335A As a Statistical Multiplexer
58	5959-9274	HPPA Pathing Conventions for HP3000 900 Series Processors - Version 2
59	5959-9289	HP 2334A and HP 2334A Configuration Recipes
60	5959-9301	TurboIMAGE's I-FILES and J-FILES
61	5959-7385	HPDeskManager - Looking Behind the Scenes
62	5959-7803	Setting Up a System Dictionary
63	5959-7834	Configuring Telesupport Modems for MPE V/E Systems
64	5960-1816	Finding Solutions in HP SupportLine
65	5960-1817	Using the Electronic Call Feature of HP SupportLine
66	5960-1818	Using the Feedback Feature of HP SupportLine
67	5960-1819	Printing Documents from HP SupportLine
68	5960-1820	HP SupportLine Commands
69	5960-2901	Nonsystem Volume Sets and the Migration of Private Volumes to an S9000 HP 3000
70	5960-2907	Modem Links for Remote Console and Standard DTC Connections on Commercial XL HPPA Systems
71	5960-2918	Asynchronous Cabling
72	5960-2919	BRW Tips and Tricks
73	5960-2998	SNA NRJE Configuration
74	5960-2999	SNA IMF Configuration
75	5060-3000	XL NRJE Configuration

HP 3000 (continued) (continued)

Note #	Part Number	Topic
76	5960-4301	XL IMF Configuration
77	5960-4302	Calling the BRW Intrinsics
78	5960-4303	PUB.SYS What Is Behind It?
79	5960-4625	Conquest of Disc Space
80	5960-4633	Looking Behind the Scenes of Resource Sharing
81	5960-4637	MPE/XL System Interrupt Recovery Procedures
82	5960-4347	Private Volumes
83	5960-4396	Serial Printer Configuration
84	5960-4334	How to Migrate FORTRAN Programs to Newer Compilers and XL Hardware
85	5960-4335	The Optimization of Programs in MPE/XL

NOTES

NOTES

NOTES

NOTES

HP 3000 Application Note Request

☐ Please send me the HP 3000 Application Notes listed below. To order an Application Note just fill in the Application Note number and the part no. in the space provided below. (Use the Published Application Notes form)

AN _/_ 59 _/_-_/_/_/_	AN _/_ 59 _/_-_/_/_/_
AN _/_ 59 _/_-_/_/_/_	AN _/_ 59 _/_-_/_/_/_
AN _/_ 59 _/_-_/_/_/_	AN _/_ 59 _/_-_/_/_/_
AN _/_ 59 _/_-_/_/_/_	AN _/_ 59 _/_-_/_/_/_
AN _/_ 59 _/_-_/_/_/_	AN _/_ 59 _/_-_/_/_/_

Business phone Extension Best time to call

Name Title/Dept.

Company Division

Mailing Address Mail Stop/Bldg./Rm.

City State Zip Code

1515

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 609 CUPERTINO, CA

POSTAGE WILL BE PAID BY ADDRESSEE

HEWLETT-PACKARD
19310 PRUNERIDGE AVE
BLDG 49 AM
CUPERTINO CA 95014-9826

Customer Comment Card

Application Note No.: ________

Part No.: ________________

We welcome your evaluation of this Application Note. Your comments and suggestions will help us improve our publications. Attach additional pages if necessary.

Please circle the following Yes or No:

■ Is the information technically accurate?	Yes	No
■ Are instructions complete?	Yes	No
■ Are concepts and wording easy to understand?	Yes	No
■ Are the examples and pictures helpful?	Yes	No
■ Is the format of this note convenient in size, arrangement and readability?	Yes	No
■ Did you receive the published application notes requested in a timely manner?	Yes	No

Additional Comments and/or suggestions for future application notes: ____________________

____

____

____

____

Please provide:

Name: ____________________ Title: ____________________

Company: ____________________ Address: ____________________

City: ______________ State: _____ Zip Code/Country ____________________

Please send to :

HEWLETT PACKARD

No postage is required. Just remove this card, fold so that the pre-addressed label is on the outside, secure and mail.

Thank you for your assistance.

Printed in U.S.A. 1990

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 95, MT. VIEW, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Learning Products Manager
Hewlett-Packard Company
100 Mayfield Avenue
Mail Stop 37MA
Attention: George Enos
Mt. View, CA 94043

FOLD FOLD

** For HP Internal Reference Only **

Manufacturing Part Number

5960-4335

Customer Order Number

NONE

Printed in USA