Porting "real" applications to OpenVMS I64

Guy Peleg
OpenVMS Systems Division
Hewlett-Packard Company
guy.peleg@hp.com

Agenda

- Porting Overview
- Conditionalized code
- IEEE Floating-Point
- Build tools
- Miscellaneous topics
- Using the XDELTA debugger
- Next steps…

Porting to OpenVMS I64

- Porting applications to I64 is easy
- Usually all that is required is to recompile/ relink and requalify the application.
  - Privileged code may require more effort
  - Porting 100,000 lines of C code did not require even one change

HOWEVER

Porting to OpenVMS I64

- MANY!!! Things have changed in the O/S
  - Different primitives
  - Different default floating point standard
  - New compilers
  - New image format
  - New calling standard
  - No console/PAL code

Most changes are transparent but these changes might affect your application

Porting to OpenVMS I64

- The purpose of this presentation is to use the experience we gained porting the base O/S, to ease the porting of your application.
  - It is a non-goal to discuss changes made to the O/S (like set predicate register x before calling the system service dispatcher)
  - We are trying to cover the most common issues. It is possible that you will encountered something not covered here.
- Might be used as a cookbook of things to look for before starting the port of your application.
- Again for most of the people….recompile and relink will do……
- All of the coding examples are working programs. We encourage you to test them yourself and get real experience…

How do I start?

- How do I start porting my application?
  - There are several approaches:
    - Re-examine the application for potential “hot spots”
    - Compile/link and see what’s broken
    - Compile and examine new messages

- There is no right approach, take the one that you feel most comfortable with

- Porting Overview
- Conditionalized code
- IEEE Floating-Point
- Build tools
- Miscellaneous topics
- Using the XDELTA debugger
- Next steps…

Conditionalized code

- This is the first (and easiest) step to take
  - Usually, IA64 should take what use to be the Alpha code path.
    - In some cases, IA64 specific code path should be added

```
#include <stdio.h>
#include <arch_defs>
void main()
{
#ifdef __vax
        printf("This is the VAX codepath");
#endif
#ifdef __alpha
        printf("This is not the VAX codepath");
}
```

Conditionalized code

- Macro

```
IF DF ALPHA           IF DF IA64
ENDC                  ENDC
```

- C

```
#ifdef __alpha        #ifdef __ia64
#endif                #endif
```

- Bliss

```
%IF ALPHA %THEN       %IF IA64 %THEN
%FI                   %FI
```

Conditionalized code – example

```
IPL31> type arch_test.c

#include <stdio.h>
#include <arch_defs>
void main()
{
#ifdef __vax
        printf("This will be printed on VAX\n");
#endif
#ifdef ALPHA
        printf("This will be printed on Alpha\n");
#endif
#ifdef __ia64
        printf("This will be printed on IA64\n");
#endif
#ifndef __vax
        printf("This program is not running on VAX");
#endif
}
```

Conditionalized code

Executed on IA64 system

```
IPL31> write sys$output f$getsyi("arch_name")
IA64
IPL31> r arch_test
This will be printed on IA64
This program is not running on VAX
IPL31>
```

Executed on Alpha system

```
MIKAXP> write sys$output f$getsyi("arch_name")
Alpha
MIKAXP> r arch_test
This will be printed on Alpha
This program is not running on VAX
```

- Porting Overview
- Conditionalized code
- IEEE Floating-Point
- Build tools
- Miscellaneous topics
- Using the XDELTA debugger
- Next steps…

IEEE floating- point

- This is one of the biggest porting issues.
- Itanium supports only IEEE floating-point in hardware
- On IA64 - IEEE floating-point is the default floating point format for the compilers.
  - VAX floating point formats will be supported when specified as a switch to the compilers
  - The compilers generate code to call conversion routines (performance hit).

IEEE floating- point

- To use IEEE on both Alpha and IPF, developers have to change the source to use S & T versions of the APIs.
  - Some functions (like sin, cos,….) already know how to handle IEEE and require no changes to the application.

Q: My application uses F-float. I'm currently porting it to I64, I'm excited enough about the new architecture and I don't want to make any source changes for now. What can I do?

A: Have no fear….HP OpenVMS engineering is here…

Let's take a look at a real example……

Convert seconds to delta time

```
$ ty float_test.c
#include <stdio.h>
#include <libdtdef.h>
#include <descrip>
#include <ssdef>

// prototypes
int lib$cvtf_to_internal_time();
int sys$fao();
int lib$put_output();
int lib$signal();

static $DESCRIPTOR (fao_desc, "Converted delta time: !%T");

main () {

 float  f1;
 unsigned long long int delta1;
 int    status;
 char   output[50]={0};
 struct dsc$descriptor_s outdesc;
 short int length;

 //init the descriptor
 outdesc.dsc$w_length  = sizeof(output);
 outdesc.dsc$a_pointer = (char *)&output;
 outdesc.dsc$b_class   = DSC$K_CLASS_S;
 outdesc.dsc$b_dtype   = DSC$K_DTYPE_T;

 f1 = 156.45;

 printf("This program converts %f seconds to delta time\n", f1);

 status = lib$cvtf_to_internal_time(&LIB$K_DELTA_SECONDS_F, &f1, &delta1);

 if (!(status&1))
        lib$signal(status);

 if ((sys$fao(&fao_desc,&length,&outdesc,&delta1))&1)
        lib$put_output(&outdesc);
}
```

Executed on Alpha:

```
AXP> cc float_test
AXP> link float_test
AXP> r float_test
This program converts 156.449997 seconds to delta time
Converted delta time: 00:02:36.44
```

Executed on IA64:

```
I64> cc float_test
I64> link float_test
I64> r float_test
This program converts 156.449997 seconds to delta time
%LIB-F-IVTIME, invalid time passed in, or computed
%TRACE-F-TRACEBACK, symbolic stack dump follows
image     module    routine               line      rel PC           abs PC
FLOAT_TEST                                     0000000000010240 0000000000010240
FLOAT_TEST                                     00000000000100D0 00000000000100D0
                                               0000000000000000 FFFFFFFF80B1A030
                                               0000000000000000 000000007AE1BEE0
```

Compiled again with the /FLOAT qualifier

```
I64> cc float_test/float=g_float
I64> link float_test
I64> r float_test
This program converts 156.449997 seconds to delta time
Converted delta time: 00:02:36.44
IPL31>
```

Note the program would fail on Alpha as well if compiled with ieee_float

```
AXP> cc float_test/float=ieee
AXP> link float_test
AXP> r float_test
This program converts 156.449997 seconds to delta time
%LIB-F-IVTIME, invalid time passed in, or computed
%TRACE-F-TRACEBACK, symbolic stack dump follows
```

```
  image     module     routine             line      rel PC           abs PC
 FLOAT_TEST  FLOAT_TEST  main              4514 0000000000000174 0000000000030174
 FLOAT_TEST  FLOAT_TEST  __main               0 0000000000000064 0000000000030064
                                              0 FFFFFFFF8025DE94 FFFFFFFF8025DE94
```

IEEE floating- point

- On IA64, the default value for the /FLOAT qualifier is IEEE_FLOAT. This program relies on the binary representation of the floating point value and therefore it fails on IA64.
- Compiled on IA64 with /FLOAT=G_FLOAT forced the compiler to use the default Alpha representation. No code changes are required in this case but there is some runtime cost.
- To use IEEE floating point representation, this program should be modified to use LIB$CVTS_TO_INTERNAL_TIME
- LIB$WAIT is another common example where floating point conversion may become an issue…let’s take a look….

```
AXP> ty wait.c
#include <stdio.h>
main()
{
float wait=7.0;

        printf("Waiting 7 seconds\n");
        lib$wait(&wait,0,0);
        printf("I'm done wainting..ciao...\n");

        return 0;
}
```

Executed on Alpha:

```
AXP> cc wait
AXP> link wait
AXP> r wait
Waiting 7 seconds
I'm done wainting..ciao...
```

Executed on I64:

```
I64> cc wait
I64> link wait
I64> r wait
Waiting 7 seconds
%SYSTEM-F-FLTINV, floating invalid operation, PC=FFFFFFFF82142760, PS=0000001B
%TRACE-F-TRACEBACK, symbolic stack dump follows
image     module    routine               line      rel PC           abs PC
LIBRTL                                         000000000016C752 FFFFFFFF82142752
LIBRTL                                         000000000020F430 FFFFFFFF821E5430
WAIT                                           0000000000010250 0000000000010250
WAIT                                           0000000000010180 0000000000010180
                                               0000000000000000 FFFFFFFF80B1A030
                                               0000000000000000 000000007AE1BEE0
```

The default floating point format used by LIB$WAIT is F_FLOAT, which does not match the default floating point format used on I64 (S_FLOAT)

Here is a modified version that will work on both platforms, using the native floating point formats

```
I64> ty wait_common.c
#include <stdio.h>
#include <arch_defs>
#include <libwaitdef>
main()
{
float wait=7.0;
#ifdef __alpha
        int mask = LIB$K_VAX_F;
#endif
#ifdef __ia64
        int mask = LIB$K_IEEE_S;
#endif
        printf("Waiting 7 seconds\n");
        lib$wait(&wait,0,&mask);
        printf("I'm done wainting..ciao...\n");

        return 0;
}
```

- Porting Overview
- Conditionalized code
- IEEE Floating-Point
- Build tools
- Miscellaneous topics
- Using the XDELTA debugger
- Next steps…

Build tools

- The port to Itanium requires that applications will be built using the latest versions of the compilers
  - Some applications being built with older versions might see some issues introduced by changes to the compilers and even bugfixes within the compilers.
  - For example - Older versions of Bliss used to return a value for functions defined NOVALUE (similar to C void)
    - On I64 this has been fixed and some utilities failed
  - You might get away with relying on uninitialized variables, but on I64 you will be severely punished
- It is recommended to build your application on Alpha, using the latest version of the compilers you are using to uncover any hidden bugs/changes. This will make the move to the new platform easier.

Example – Moving from F77 to F90

- When using double precision float (REAL*8) for doing direct assignment (a=5.3)

F77 uses double precision

F90 uses single precision. The result is slightly further away from the real 5.3 value.

- A computation will produce a different result with no error signaled.
- Possible solutions:
  - Fix the coding bug, as the assignment is wrong.
    - Change the assignment to a=5.3D0 or a=5.3_8
    - 5.3D0 works for both F77 and F90
  - Compile using the /ASSUME=FP_CONSTANT switch

Example – Moving from F77 to F90

```
IPL31> ty float.for
       REAL*8             TEST

       TEST = 5.3
       PRINT 100,TEST
 100   FORMAT(F,' assigned as TEST = 5.3 ')

       TEST = 5.3D0
       PRINT 200,TEST
 200   FORMAT(F,' assigned as TEST = 5.3D0')

       END
IPL31> for float
IPL31> link float
IPL31> r float
      5.3000001907348633 assigned as TEST = 5.3
      5.2999999999999998 assigned as TEST = 5.3D0
IPL31> for/assume=fp_const float
IPL31> link float
IPL31> r float
      5.2999999999999998 assigned as TEST = 5.3
      5.2999999999999998 assigned as TEST = 5.3D0
```

- Porting Overview
- Conditionalized code
- IEEE Floating-Point
- Build tools
- Miscellaneous topics
- Using the XDELTA debugger
- Next steps…

Improperly declared functions and data

- C function declarations that points to objects that are not functions, may work on Alpha but these declarations will not work on IA64
  - Also seen with Bliss and Fortran

- The problem may manifest itself in many ways
  - For whatever reason, the most common symptom is a call to routine CLI$DCL_PARSE that fails with CLI-E-INVTAB

```
external int my_cld();
  status = cli$dcl_parse(&cmdstr, my_cld,
                         lib$get_input, lib$get_input);
external int my_cld;
  status = cli$dcl_parse(&cmdstr, &my_cld,
                         lib$get_input, lib$get_input);
```

Improperly declared functions and data

- We have also seen LIB$TABLE_PARSE failing with syntax error when the state table and/or key table addresses were declared as a function

```
VMS> diff src$:SJ_PARSE_DESCRIP.C;3
  ************
  File WORK4:[SRC]SJ_PARSE_DESCRIP.C;3
  18 extern int parse_state, parse_key; /* parse table */
  19
  ******
  File WORK4:[SRC]SJ_PARSE_DESCRIP.C;2
  18 extern int parse_state(), parse_key(); /* parse table */
  19
  ************
  ************
  File WORK4:[SRC]SJ_PARSE_DESCRIP.C;3
  51 status = lib$table_parse(&tpablk, &parse_state, &parse_key);
  52
  ******
  File WORK4:[SRC]SJ_PARSE_DESCRIP.C;2
  51 status = lib$table_parse(&tpablk, parse_state, parse_key);
  52
  ************

  Number of difference sections found: 2
  Number of difference records found: 2

  DIFFERENCES /IGNORE=()/MERGED=1-
  WORK4:[SRC]SJ_PARSE_DESCRIP.C;3-
  WORK4:[SRC]SJ_PARSE_DESCRIP.C;2
```

Improperly declared functions and data

- The Fortran variant of the fix would be

```
      CDEC$ ATTRIBUTES EXTERN :: MY_CLD
```

- Lately, the Linker has been modified to detect this condition

```
%ILINK-I-DIFTYPE, symbol TEST_CLD of type OBJECT cannot be referenced as type FUNC
   module: TEST
   file: $1$DKC600:[IA64]TEST.OBJ;2
```

Hardware Model

- The hardware model for all IA64 systems is set to 4096
  - HW model is set to 0 in E8.1 and E8.2

```
I64> write sys$output f$getsyi("hw_name")
HP rx2600  (900MHz/1.5MB)
I64> write sys$output f$getsyi("hw_model")
4096
```

- Any code relying on the hardware model of the system has to change.
- SHOW MEMORY used to determine the page size based on the following algorithm:

```
if (hardware_model>=1024)
      page_size=8192;
else page_size=512;
```

- SHOW MEMORY has been modified to use the page_size item code of $GETSYI system service on all three architectures.

IMACRO

- On I64 the calling standard changed
  - We now use Intel’s software conventions
  - IA64 only preserves register R4-R7 across routine calls (Alpha preserves R2-R15)
  - For programs written in high-level languages (like C,Bliss) this difference is not visible.
  - When MACRO-32 programs added you have to start worrying about how to pass register parameters between languages.
  - High-level languages might trash a register IMACRO assumed to be preserved (and vice versa)
  - Please reference the IMACRO porting guide for more details

IMACRO - coding changes

- JSB_ENTRYs that reference R16-R21should be changed to .CALL_ENTRYs that reference n(AP)

```
 XFC_COMPARE_QIOS:
.JSB_ENTRY
 MOVL    (R16),R0
 MOVL    (R17),R1
 EVAX_SUBQ CFS$Q_TOTALQIOS(R0),-
           CFS$Q_TOTALQIOS(R1),R0
 RSB
```

```
XFC_COMPARE_QIOS:
.CALL_ENTRY
MOVL    @4(ap),R0
MOVL    @8(ap),R1
EVAX_SUBQ CFS$Q_TOTALQIOS(R0),-
          CFS$Q_TOTALQIOS(R1),R0
RET
```

- Code that is moving data in R16-R21 and then perform a JSB should be modified to use $SETUP_CALL64 and $CALL64

IMACRO - coding changes

- MACRO32 JSBs to any other language (Bliss/C) routines
  - If IMACRO can't determine the language of a target JSB, the following message will be generated:

```
JSB     (R6)  ;ALLOCATE AND INSERT ENTRY IN SYMBOL TABLE
%IMAC-I-CODGENINF, (1) Indirect JSB with default linkage
```

– .USE_LINKAGE directive with the LANGUAGE=OTHER option tells iMacro that the target routine is written in a language other than IMACRO. IMACRO will than save and restore R2,R3,R8-R15 around the JSB except for registers in the output mask.

It is recommended to add a USE_LINKAGE statement prior to the JSB call

.use_linkage language=other (or language=macro if the target routine is in MACRO)
JSB (R6)

IMACRO - coding changes

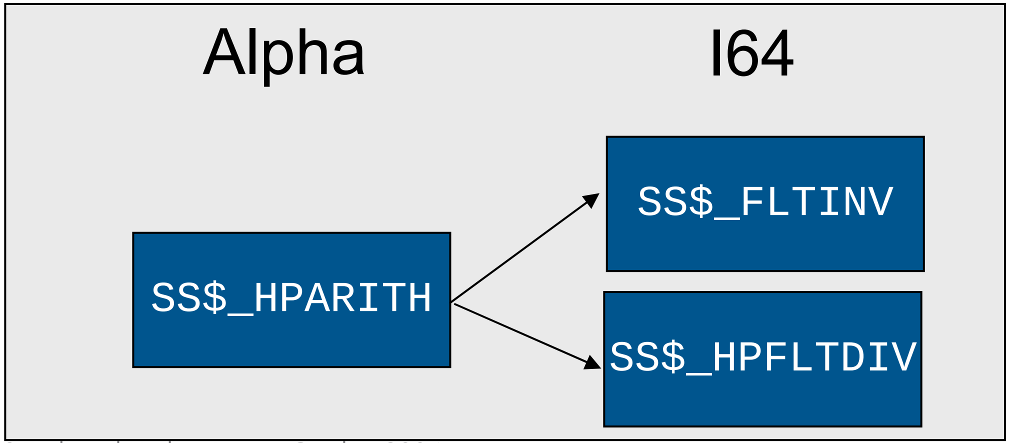
- MACRO32 CALL/CALLG to non-standard routines
  - A non standard routine (written in Bliss C or MACRO) returns a value in a register other than R0 or R1
  - Since IMACRO saves and restores R2,R3,R8-R15, the returned value may be overridden
  - .CALL_LINKAGE or .USE_LINKAGE should be used in every module that calls the non standard routine.
  - For example,
    .CALL_LINKAGE RTN_NAME=FOO$BAR, OUTPUT=<R3,R8,R10>
  - The call_linkage needs only to appear once in every module
  - The .USE_LINKAGE directive will be used only once
  - Here is a small example from DCL, where a MACRO routine is calling a C routine.

```
.IF DF IA64
.use_linkage input=<r0,r1,r2,r3,r8,r9>, output=<r0,r1,r2>,
             language=other
.ENDC
 JSB    DCL$FID_TO_NAME         ; dispatch to the action routine
```

Condition Handlers Use of SS$_HPARITH

On OpenVMS Alpha, SS$_HPARITH is signaled for a number of arithmetic error conditions. On OpenVMS I64, SS$_HPARITH is never signaled for arithmetic error conditions; instead, the more specialized SS$_FLTINV and SS$_FLTDIV error codes are signaled on OpenVMS I64.

Update condition handlers to detect these more specialized error codes. In order to keep code common for both architectures, wherever the code refers to SS$_HPARITH, extend it for OpenVMS I64 to also consider SS$_FLTINV and SS$_FLTDIV.



Up yours!

Quotas and process settings

- OpenVMS I64 images are much larger, sometimes 3x-4x!
- Ensure your pgflquo and bytlm are (at least) 4x-10x your Alpha settings.
  - $ set default sys$system
  - $ run authorize
  - UAF> mod your_account/pgflquo=nnnnnn
  - UAF> mod your_account/bytlm=nnnnnn

THREADS

- THREADCP tool was not ported to OpenVMS I64
  - Relink to change threads related characteristics of an image
  - Use the new SET IMAGE command

- If your application increases the stack size for a thread from the default size, you should increase it more

  HP recommends starting with an increase of three 8-Kb pages (24576 bytes).

There is more……

- The topics covered so far are the most common issues seen by people porting their applications. Here is a list of less common (and much more complicated) issues.
- We adopted Intel's calling standard. Code with knowledge about the calling standards will have to change
  - Stack/frame walking – the code will need to be modified to use the new LIB$*_INVO_* routines
  - Home grown stack switching/threading – the code will need to be ported to use KPs
- We adopted the ELF/DWARF formats. Code with knowledge about image format and debug format will have to change
  - Calling LIB$FIND_IMAGE_SYMBOL and friends does not count. The LIB$ routines were modified to support the new formats

Reading EXE and OBJ files

- Use ANALYZE/IMAGE vs. parsing the EXE file.
- We might provide a callable interface into SHOW/SET image.

ANALYZE/IMAGE	DCL Symbol that is set	Sample Value
/SELECT=ARCHITECTURE	ANALYZE$ARCHITECTURE	OpenVMS IA64
/SELECT=NAME	ANALYZE$NAME	"DECC$COMPILER"
/SELECT=IDENTIFICATION=IMAGE	ANALYZE$IDENTIFICATION	"C T7.1-003"
/SELECT=IDENTIFICATION=LINKER	ANALYZE$LINKER_IDENTIFICATION	"Linker I02-08"
/SELECT=LINK_TIME	ANALYZE$LINK_TIME	"6/29/2004 4:26:35 PM"
/SELECT=FILE_TYPE	ANALYZE$FILE_TYPE	Image
/SELECT=IMAGE_TYPE	ANALYZE$IMAGE_TYPE	Executable

- Porting Overview
- Conditionalized code
- IEEE Floating-Point
- Build tools
- Miscellaneous topics
- Using the XDELTA debugger
- Next steps…

Debugging using XDELTA

- The system must be booted with XDELTA.
- From the EFI shell
  - SHELL> SET VMS_FLAGS “0,3” (bit 1 should be set)
- The SYSGEN parameter BREAKPOINTS must be set to allow breaking in user,super or exec mode

Use with Caution!

- Add breakpoints to your code
  - Macro

    ```
    ia64_break      #break$c_dbg_inibrk
    ```
  - C

    ```
    __break(BREAK$C_DBG_INIBRK);
    ```

```
I64> r ast
Brk 0 at 00010030 on CPU 0
00010030!              break.m     080003  (New IPL = 0)  (New mode = USER)
00010031!              add         r12 = 3FFC, r12 ;P
```

Have fun……you might want to boot with VAXCLUSTER set to 0 to prevent a clustered system from crashing with CLUEXIT

- Porting Overview
- Conditionalized code
- IEEE Floating-Point
- Build tools
- Miscellaneous topics
- Using the XDELTA debugger
- Next steps…

Alignment faults

- Once the port of the application has been completed, you might want to look at alignment faults
  - Alignment faults are expensive on Alpha but are 100 times more expensive on IA64
  - The DEBUG SET MODULE/ALL command used to take 90 seconds. After fixing some alignment faults, it now takes 2 seconds.
  - DCL procedures take approx. 10% less time to execute after fixing alignment faults in DCL.

  - You may detect alignment faults using FLT extension in SDA or using SET BREAK/ALIGN option in the debugger
  - Some alignment faults are easy to fix, some are very hard and some are close to impossible.

Acknowledgements

- The following people were kind enough to share their experience with me and made this presentation possible:
  - Dave Sweeney
  - Anders Johansson
  - John Reagan
  - Jeff Nelson
  - Christian Moser

hp
invent

Integrity Servers – Hardware Overview

- No “Vax like” or “Alpha like” console
- Has multiple consoles:
  - Management Processor (MP)
  - Baseboard Management Console (BMC)
  - Both attempt to be common across the entire hardware range
- Uses Extensible Firmware Interface (EFI) rather than BIOS.

MP console

- Runs with box level power, even with system off.
- Local, remote (modem) and network connectivity
- Console configuration (terminal type, etc.)
- Network configuration (hostname, IP address, etc.)
- Multiple console sessions (one writer, many readers)
- Provides ability to copy files over the netwrok (firmware updates)

BMC

- Runs with main board powered up
- Local connectivity (9 pin serial)
- Power up, self tests
- Device detection
- Console configuration
- No graphics console

EFI

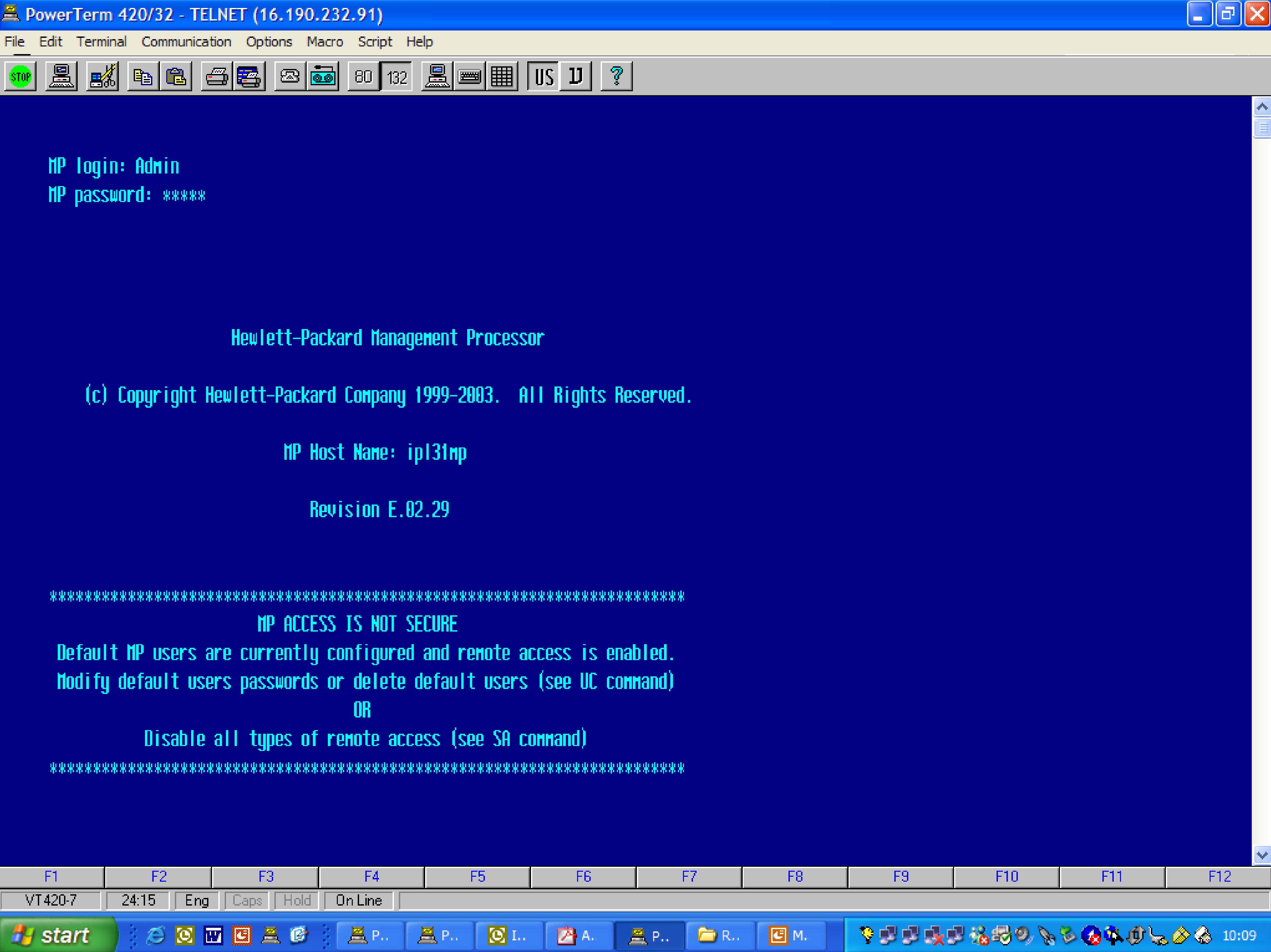
- Mini operating system
- FAT formatted file system (FAT12, FAT16 and FAT32), VMS presents FAT16 partition to EFI
- Boot menu and defaults
- Environment variables (VMS_FLAGS, etc.)
- VMS_LOADER.EFI finds and loads IPB.EXE
- IPB.EXE understands the OpenVMS file system, EFI does not.

Boot and Device detection

- EFI boot loader from FAT partition (hidden as a container file on the system disk)
- Boot flags passed through environment variables
- Reads executive into memory
- Passes control to the executive
- The system uses ACPI (Advanced Configuration and Power Interface) for device detection by the firmware
- Devices appear as a set of CSRs (Control and Status Registers) in physical memory – the I/O space.

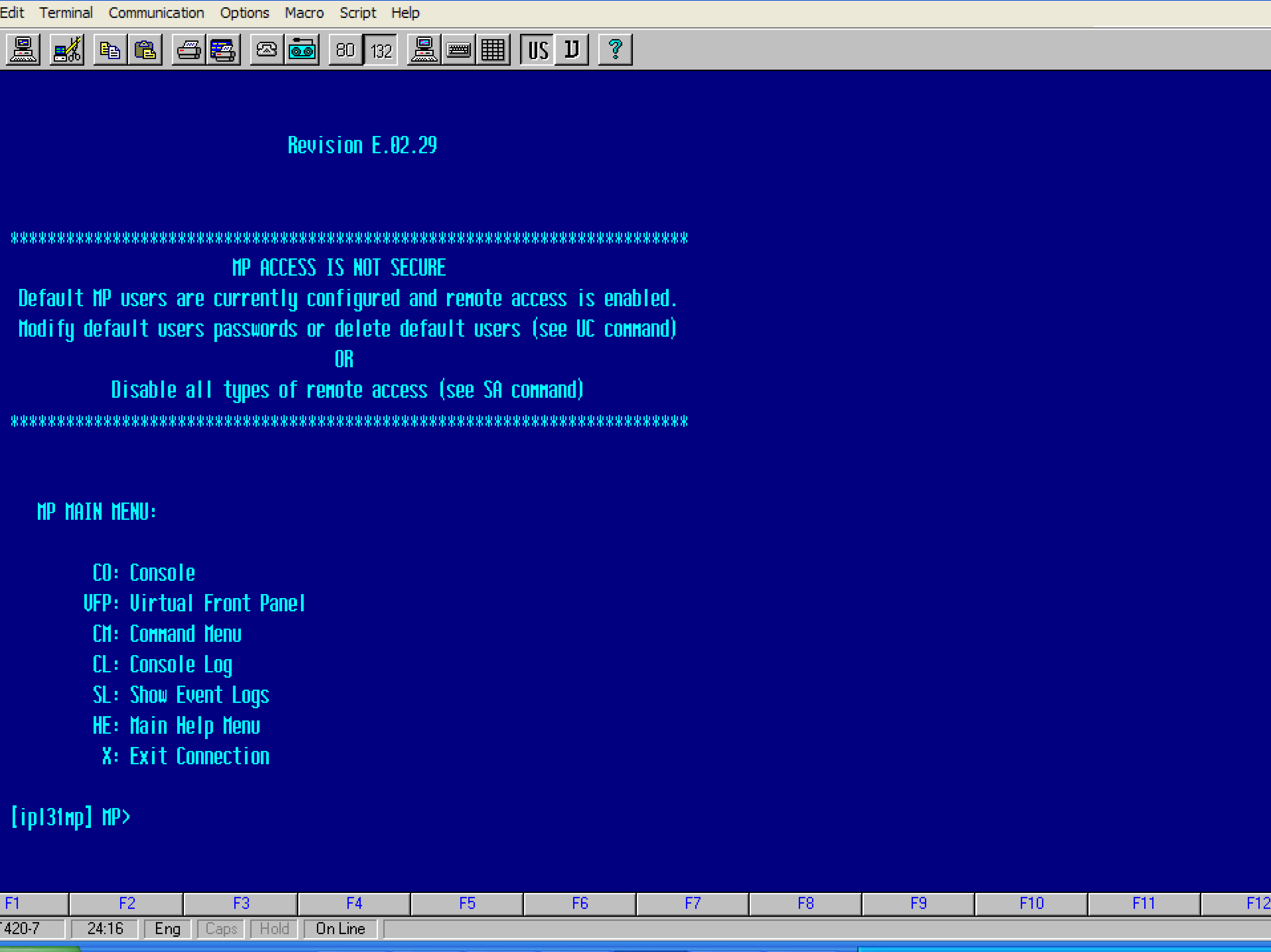
Boot and Device detection

- Devices have interrupt vectors which connect a device interrupt request to the device driver interrupt service routine. Device data obtained from ACPI data.
- ACPI data indicates device type.
- SYSMAN IO AUTO will query ACPI data to find devices and set up OpenVMS device drivers to communicate with the hardware
- ***Now Let's take a look, how the past 6 slides look at real life....***

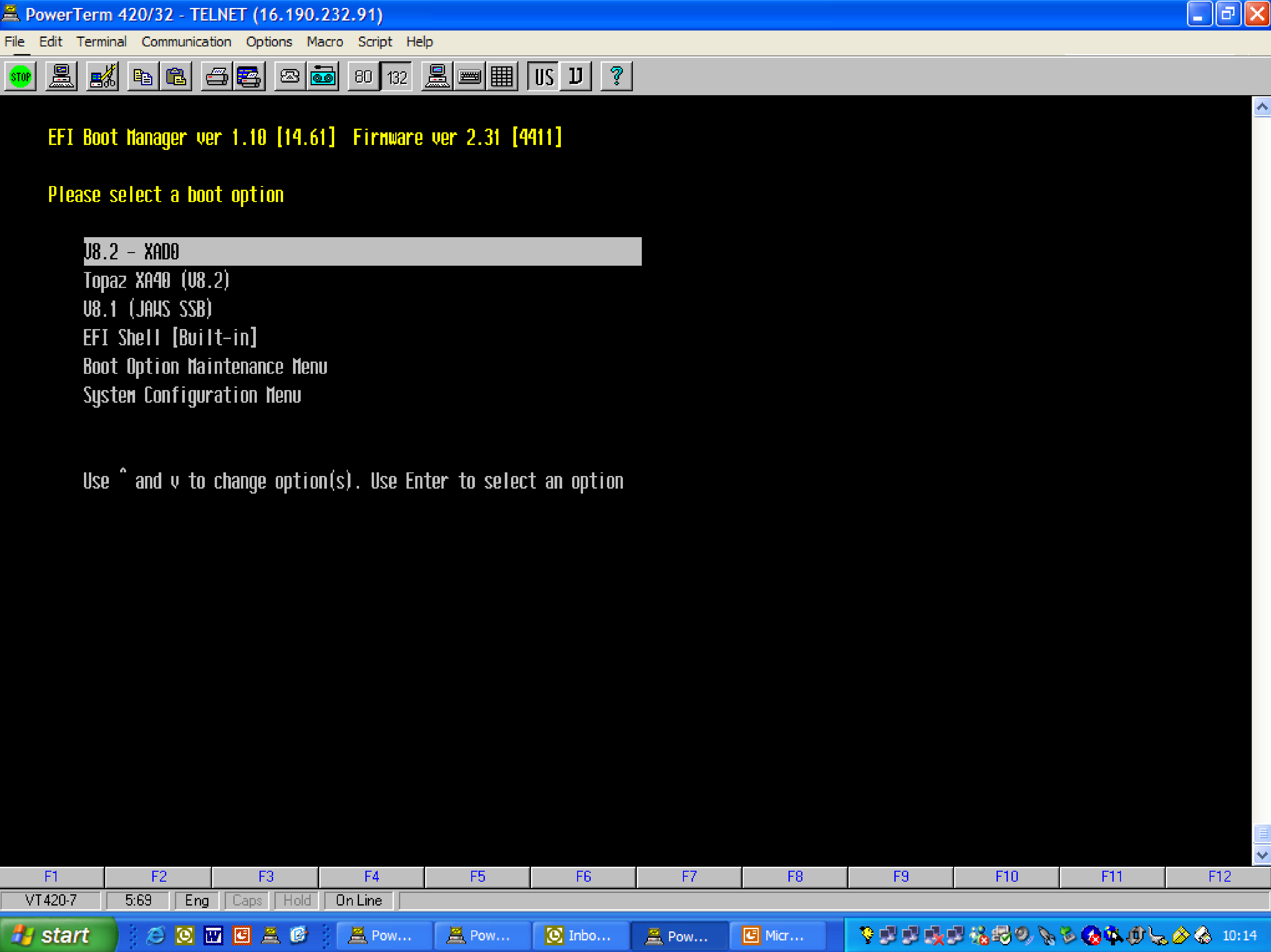


PowerTerm 420/32 - TELNET (16.190.232.91)
File Edit Terminal Communication Options Macro Script Help
STOP
80
132
US
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
VT420-7
24:15
Eng
Caps
Hold
On Line
start
10:09

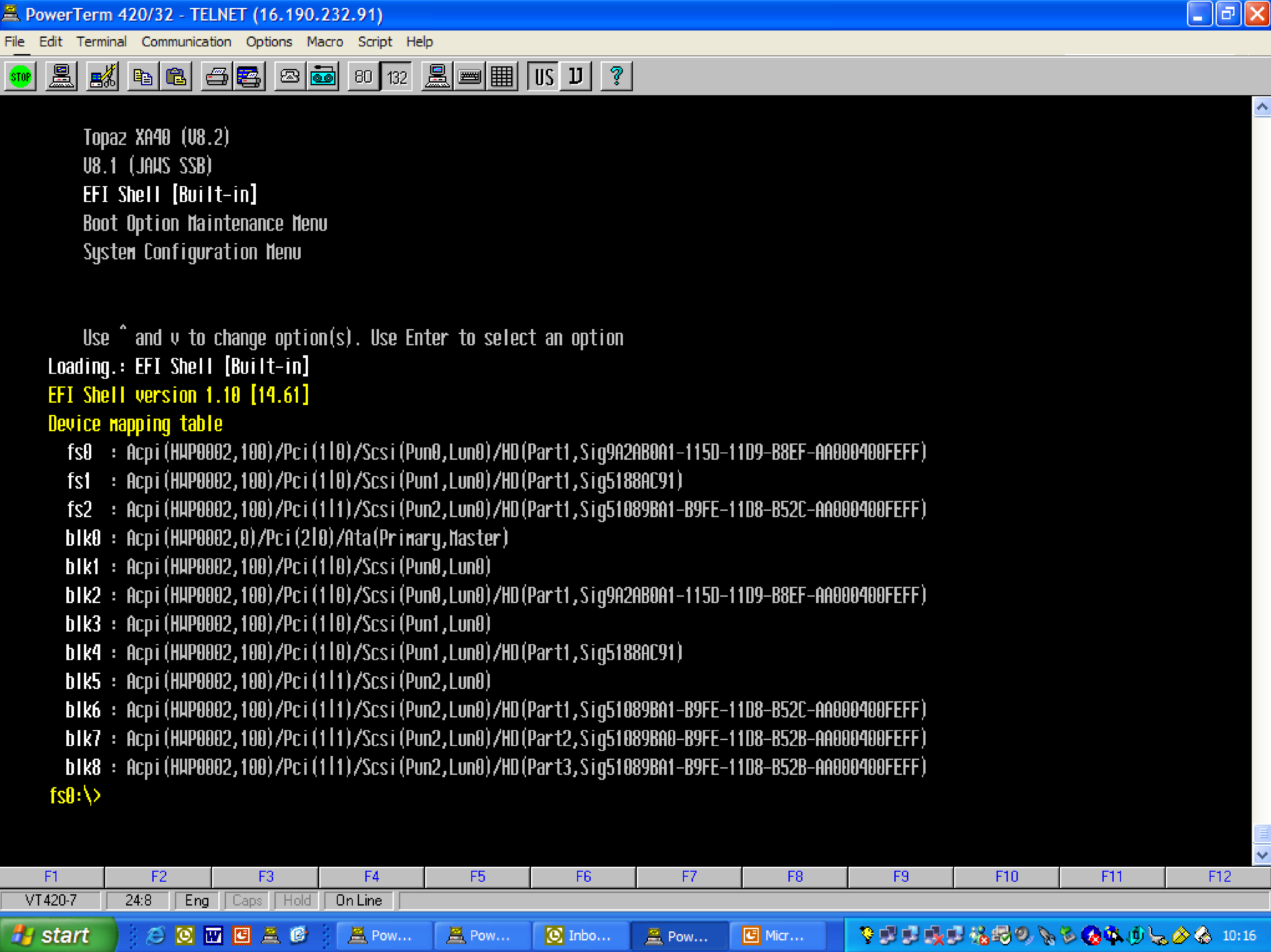
```
MP login: Admin
MP password: *****




                    Hewlett-Packard Management Processor

    (c) Copyright Hewlett-Packard Company 1999-2003.  All Rights Reserved.

                          MP Host Name: ipl31mp

                             Revision E.02.29


**************************************************************************
                      MP ACCESS IS NOT SECURE
 Default MP users are currently configured and remote access is enabled.
 Modify default users passwords or delete default users (see UC command)
                                   OR
          Disable all types of remote access (see SA command)
**************************************************************************
```



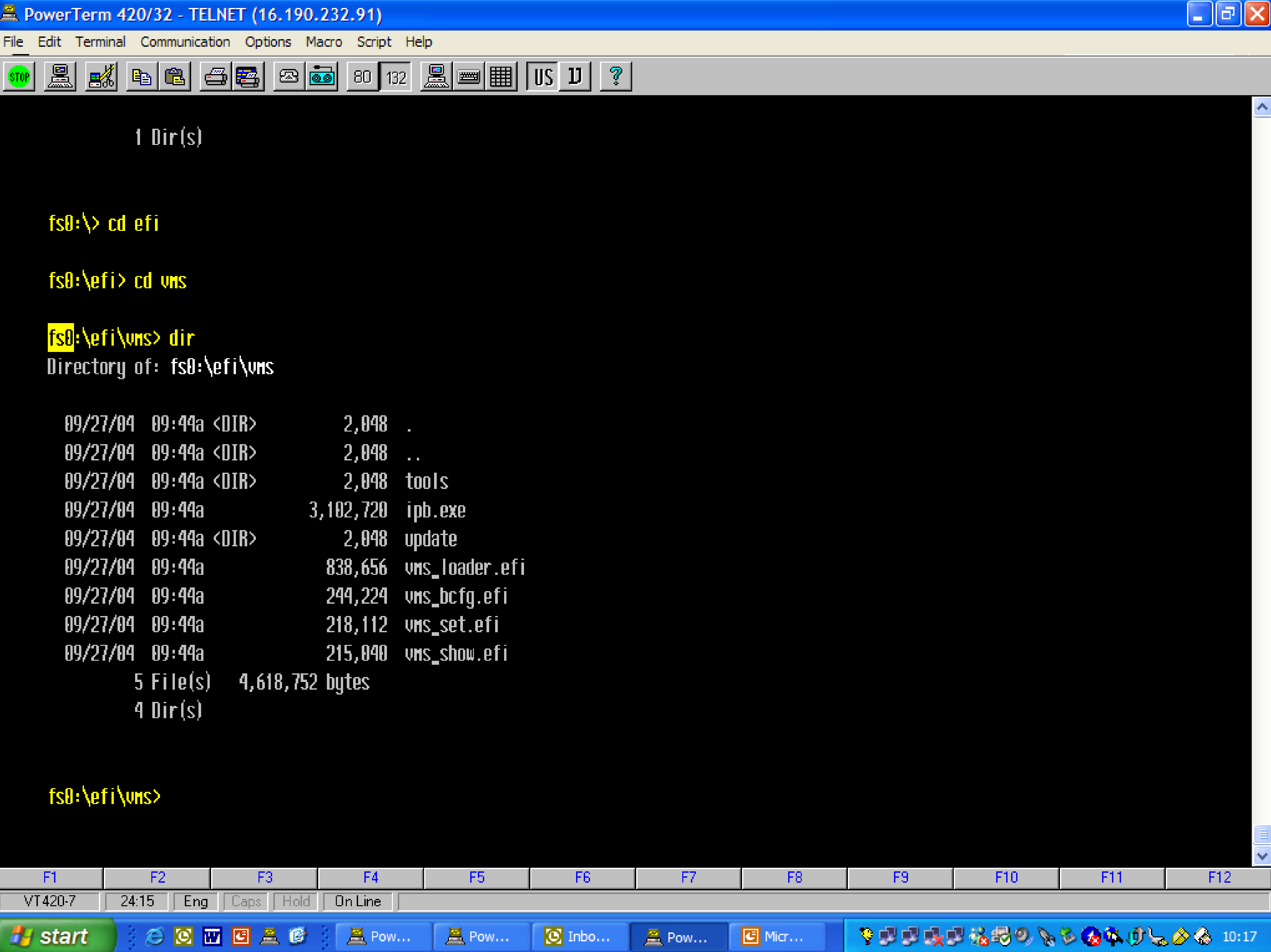
```
                              Revision E.02.29



 *************************************************************************
                         MP ACCESS IS NOT SECURE
  Default MP users are currently configured and remote access is enabled.
  Modify default users passwords or delete default users (see UC command)
                                    OR
           Disable all types of remote access (see SA command)
 *************************************************************************



    MP MAIN MENU:

          CO: Console
         VFP: Virtual Front Panel
          CM: Command Menu
          CL: Console Log
          SL: Show Event Logs
          HE: Main Help Menu
           X: Exit Connection

 [ipl31mp] MP>
```



```
   EFI Boot Manager ver 1.10 [14.61]  Firmware ver 2.31 [4411]

   Please select a boot option

       V8.2 - XAD0
       Topaz XA40 (V8.2)
       V8.1 (JAWS SSB)
       EFI Shell [Built-in]
       Boot Option Maintenance Menu
       System Configuration Menu


       Use ^ and v to change option(s). Use Enter to select an option
```

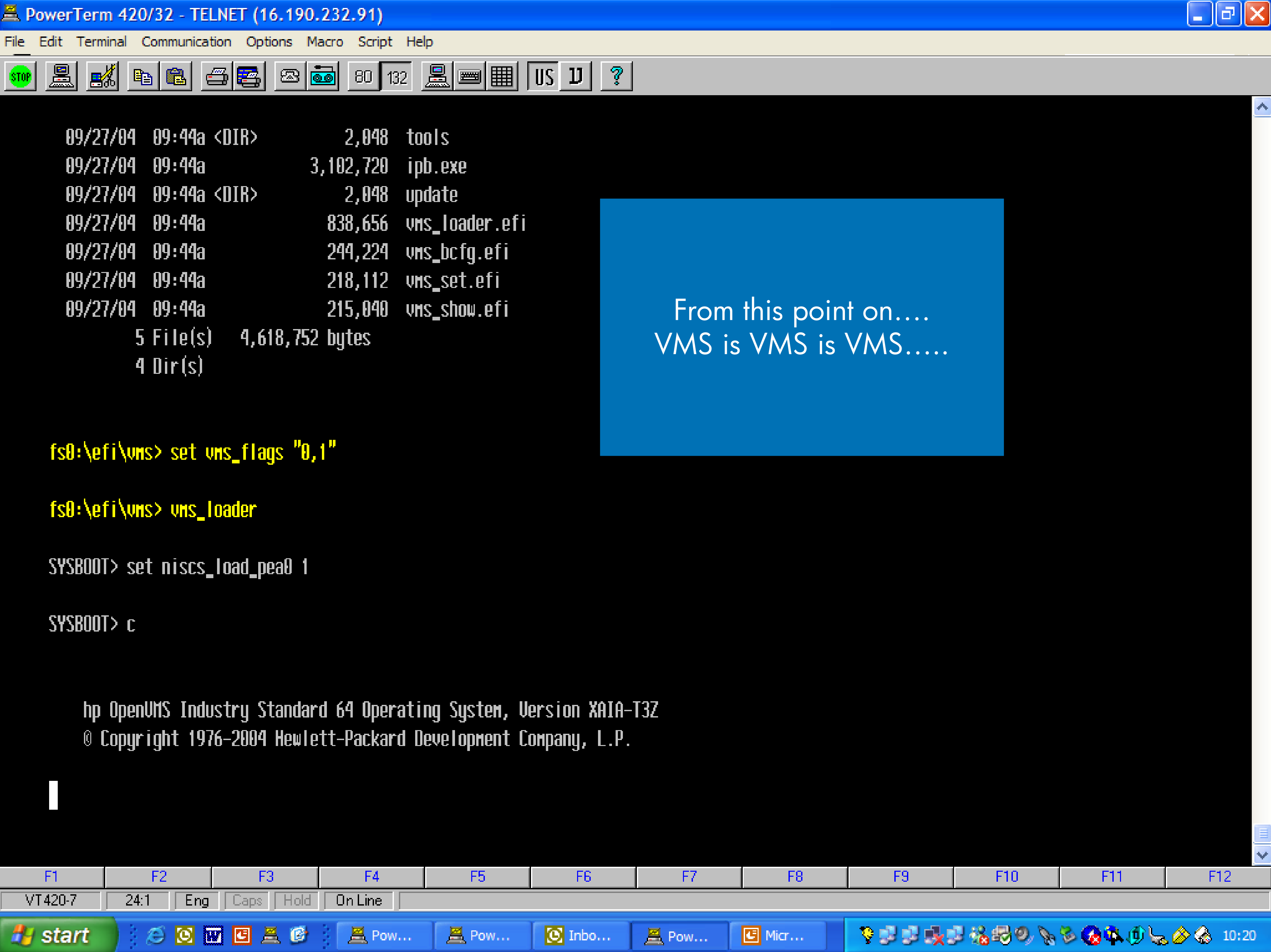

PowerTerm 420/32 - TELNET (16.190.232.91)
File Edit Terminal Communication Options Macro Script Help
STOP
80
132
US
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
VT420-7
24:8
Eng
Caps
Hold
On Line
start
Pow…
Pow…
Inbo…
Pow…
Micr…
10:16

```
     Topaz XA40 (V8.2)
     V8.1 (JAWS SSB)
     EFI Shell [Built-in]
     Boot Option Maintenance Menu
     System Configuration Menu


     Use ^ and v to change option(s). Use Enter to select an option
  Loading.: EFI Shell [Built-in]
  EFI Shell version 1.10 [14.61]
  Device mapping table
    fs0  : Acpi(HWP0002,100)/Pci(1|0)/Scsi(Pun0,Lun0)/HD(Part1,Sig9A2AB0A1-115D-11D9-B8EF-AA000400FEFF)
    fs1  : Acpi(HWP0002,100)/Pci(1|0)/Scsi(Pun1,Lun0)/HD(Part1,Sig5188AC91)
    fs2  : Acpi(HWP0002,100)/Pci(1|1)/Scsi(Pun2,Lun0)/HD(Part1,Sig51089BA1-B9FE-11D8-B52C-AA000400FEFF)
    blk0 : Acpi(HWP0002,0)/Pci(2|0)/Ata(Primary,Master)
    blk1 : Acpi(HWP0002,100)/Pci(1|0)/Scsi(Pun0,Lun0)
    blk2 : Acpi(HWP0002,100)/Pci(1|0)/Scsi(Pun0,Lun0)/HD(Part1,Sig9A2AB0A1-115D-11D9-B8EF-AA000400FEFF)
    blk3 : Acpi(HWP0002,100)/Pci(1|0)/Scsi(Pun1,Lun0)
    blk4 : Acpi(HWP0002,100)/Pci(1|0)/Scsi(Pun1,Lun0)/HD(Part1,Sig5188AC91)
    blk5 : Acpi(HWP0002,100)/Pci(1|1)/Scsi(Pun2,Lun0)
    blk6 : Acpi(HWP0002,100)/Pci(1|1)/Scsi(Pun2,Lun0)/HD(Part1,Sig51089BA1-B9FE-11D8-B52C-AA000400FEFF)
    blk7 : Acpi(HWP0002,100)/Pci(1|1)/Scsi(Pun2,Lun0)/HD(Part2,Sig51089BA0-B9FE-11D8-B52B-AA000400FEFF)
    blk8 : Acpi(HWP0002,100)/Pci(1|1)/Scsi(Pun2,Lun0)/HD(Part3,Sig51089BA1-B9FE-11D8-B52B-AA000400FEFF)
  fs0:\>
```



PowerTerm 420/32 - TELNET (16.190.232.91)
File Edit Terminal Communication Options Macro Script Help
STOP 80 132 US
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
VT420-7 24:15 Eng Caps Hold On Line
start Pow… Pow… Inbo… Pow… Micr… 10:17

```
        1 Dir(s)



fs0:\> cd efi


fs0:\efi> cd vms


fs0:\efi\vms> dir
Directory of: fs0:\efi\vms

  09/27/04  09:44a <DIR>          2,048  .
  09/27/04  09:44a <DIR>          2,048  ..
  09/27/04  09:44a <DIR>          2,048  tools
  09/27/04  09:44a            3,102,720  ipb.exe
  09/27/04  09:44a <DIR>          2,048  update
  09/27/04  09:44a              838,656  vms_loader.efi
  09/27/04  09:44a              244,224  vms_bcfg.efi
  09/27/04  09:44a              218,112  vms_set.efi
  09/27/04  09:44a              215,040  vms_show.efi
          5 File(s)   4,618,752 bytes
          4 Dir(s)



fs0:\efi\vms>
```



```
    09/27/04  09:44a <DIR>          2,048  tools
    09/27/04  09:44a            3,102,720  ipb.exe
    09/27/04  09:44a <DIR>          2,048  update
    09/27/04  09:44a              838,656  vms_loader.efi
    09/27/04  09:44a              244,224  vms_bcfg.efi
    09/27/04  09:44a              218,112  vms_set.efi
    09/27/04  09:44a              215,040  vms_show.efi
            5 File(s)    4,618,752 bytes
            4 Dir(s)


fs0:\efi\vms> set vms_flags "0,1"

fs0:\efi\vms> vms_loader

SYSBOOT> set niscs_load_pea0 1

SYSBOOT> c


    hp OpenVMS Industry Standard 64 Operating System, Version XAIA-T3Z
    © Copyright 1976-2004 Hewlett-Packard Development Company, L.P.
```

From this point on….
VMS is VMS is VMS…..

Conditionalized code
Sample Fortran 90 program

COM file

```
$!
$! Note: F90 not available on VAX
$!
$ if f$getsyi("ARCH_NAME") .EQS.
  "IA64"
$ then
$      f90/define=IA64 archdef_for
$ else
$ if f$getsyi("ARCH_NAME") .EQS.
  "Alpha"
$ then
$       f90/define=ALPHA
  archdef_for
$ endif
$ endif
$ endif
$ link archdef_for
```

Language file

```
 program archdef
        implicit none
!DEC$ IF DEFINED (VAX)
        type *, 'Running on VAX hardware'
!DEC$ ELSEIF DEFINED (ALPHA)
        type *,'Running on Alpha hardware'
!DEC$ ELSEIF DEFINED (IA64)
        type *,'Running on Integrity hardware'
!DEC$ ENDIF
        end
```

Conditionalized code
Sample Basic program

COM file

```
$!! if you VAX or Alpha system is older,
   ARCH_NAME may not be accepted
$!!   by f$getsyi... ARCH_TYPE (1-VAX,
   2=Alpha, 3=IA64) will be...
$ open/write out
   sys$disk:[]archdef.basic_include
$ write out "%LET %ARCH_TYPE =
   ",f$getsyi("arch_type")
$ close out
$ purge sys$disk:[]archdef.basic_include
$ basic archdef_bas
$ link archdef_bas
$ exit
```

Language file

```
  !
  !
     %INCLUDE "sys$disk:[]archdef.basic_include"
     program archdef_bas

     %IF (%ARCH_TYPE = 1)
         %THEN Print "Running on VAX"
         %ELSE %IF (%ARCH_TYPE = 2)
             %THEN Print "Running on Alpha"
             %ELSE  %IF (%ARCH_TYPE = 3)
               %THEN Print "Running on
   Integrity"
             %END %IF
         %END %IF
     %END %IF

end program
```

Conditionalized code
Sample Cobol program

COM file

```
$!
$ if f$getsyi("ARCH_NAME") .EQS. "IA64"
$ then
$       cobol/conditional=I archdef_cob
$ else
$ if f$getsyi("ARCH_NAME") .EQS. "VAX"
$ then
$       cobol/conditional=V archdef_cob
$ else
$ if f$getsyi("ARCH_NAME") .EQS. "Alpha"
$ then
$       cobol/conditional=A archdef_cob
$ endif
$ endif
$ endif
```

Language file

```
identification division.
program-id. HW.
environment division.
data division.
procedure division.
p1.     display "Hello World".
\A      display "Running on Alpha".
\V      display "Running on VAX".
\I      display "Running on Integrity".

        stop run.
```

Conditionalized code Sample Pascal program

COM file

```
$ pascal archdef_pas
$ link archdef_pas
```

Language file

```
program example(output);

       %if %arch_name = "Alpha"
       %then
        var handle : integer := 0;
       %elif %arch_name = "IA64"
       %then
        var handle : integer64 := 0;
       %elif %arch_name = "VAX"
       %then
        var handle : integer := 0;
       %endif

       begin
       writeln('Program running on ',%system_name,
        ' ',%arch_name,
        ' ',%system_version);

       %if %arch_name = "Alpha"
       %then
        writeln('Running on Alpha');
       %elif %arch_name = "IA64"
       %then
        writeln('Running on Integrity');
       %elif %arch_name = "VAX"
       %then
        writeln('Running on VAX');
       %endif
      end.
```