The HP OpenVMS Itanium® Calling Standard

DECUS–München 2004
1A07

John Covert

Andy Goldstein
OpenVMS Engineering

20. April 2004

What's the Problem?

An example:

On VAX and Alpha

- R0 = function return value

On Itanium® architecture

- R0 = zero

What's a Calling Standard?

In C

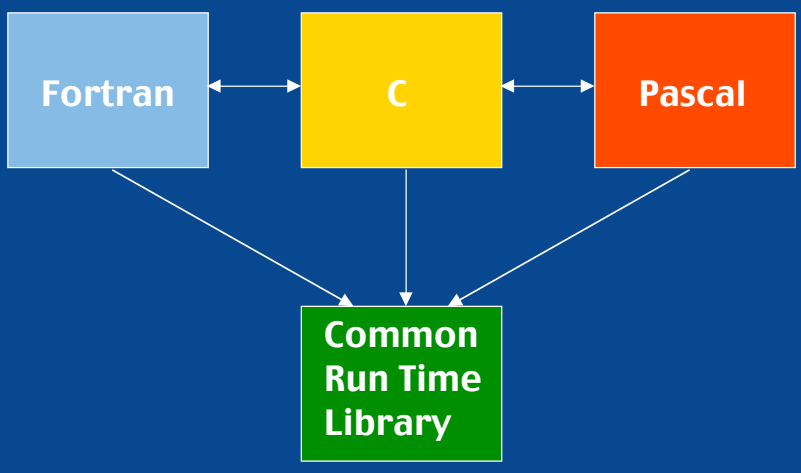
Z = func (X, Y);

Binary representation of subroutine linkage:

- Where are my arguments?
- Where's my execution environment?
- How do I get back to my caller?
- How do exceptions get handled?

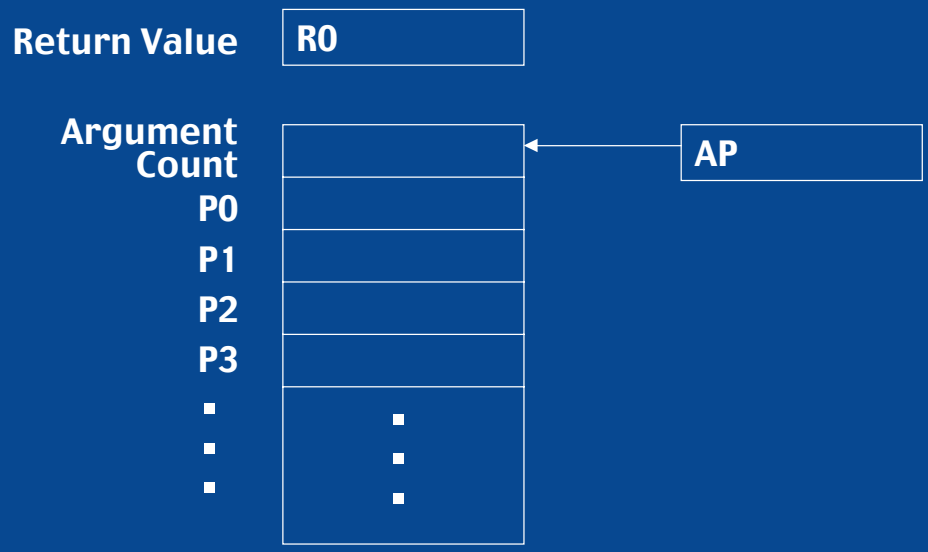
Why Have a Calling Standard?

Multi-language interoperable programming environment



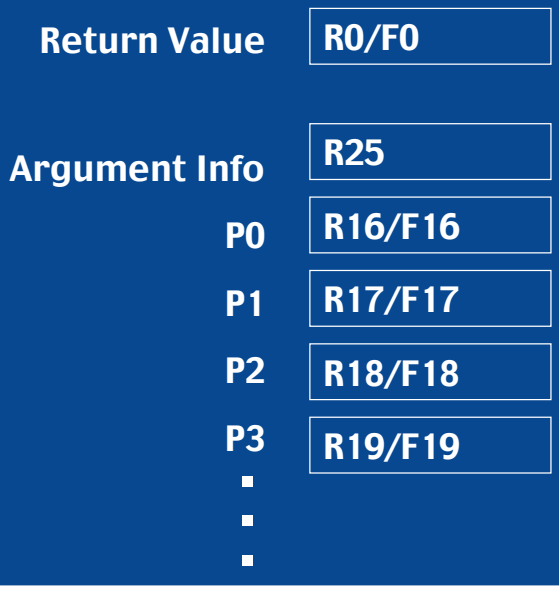
The Past

VAX – Argument List & Return Value



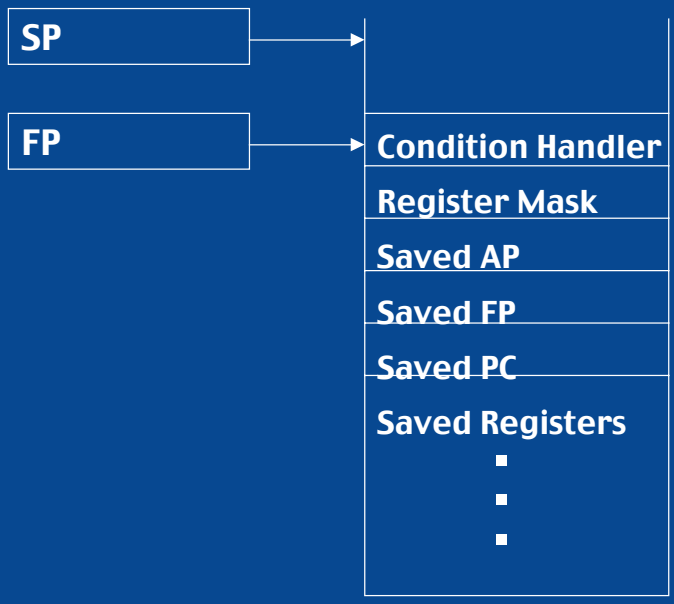
The Past

Alpha – Argument List & Return Value



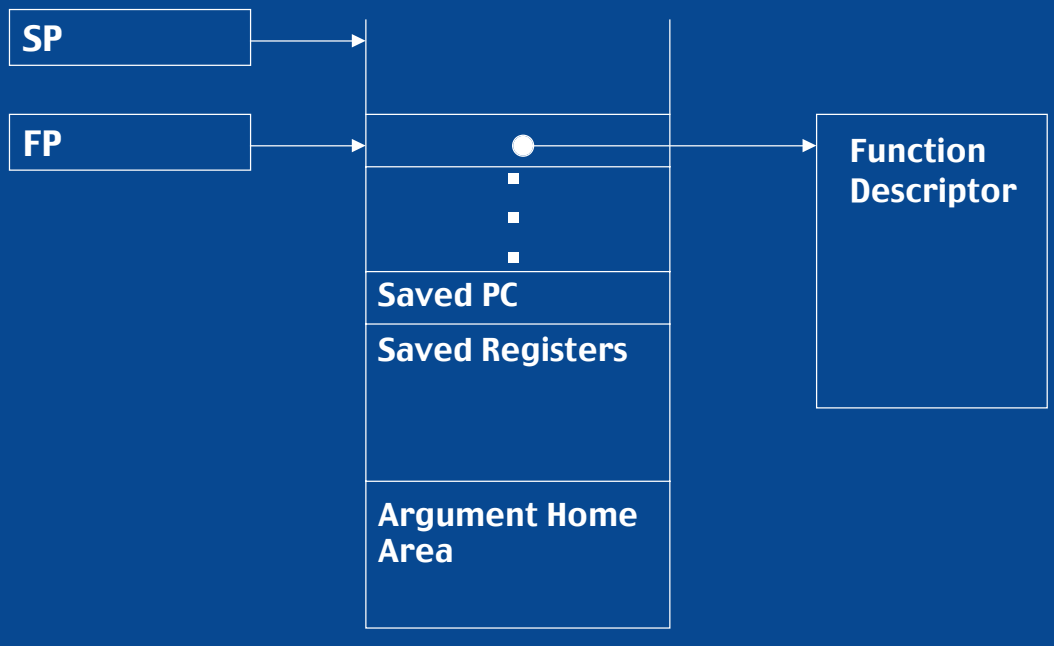
The Past

VAX – Call Frame



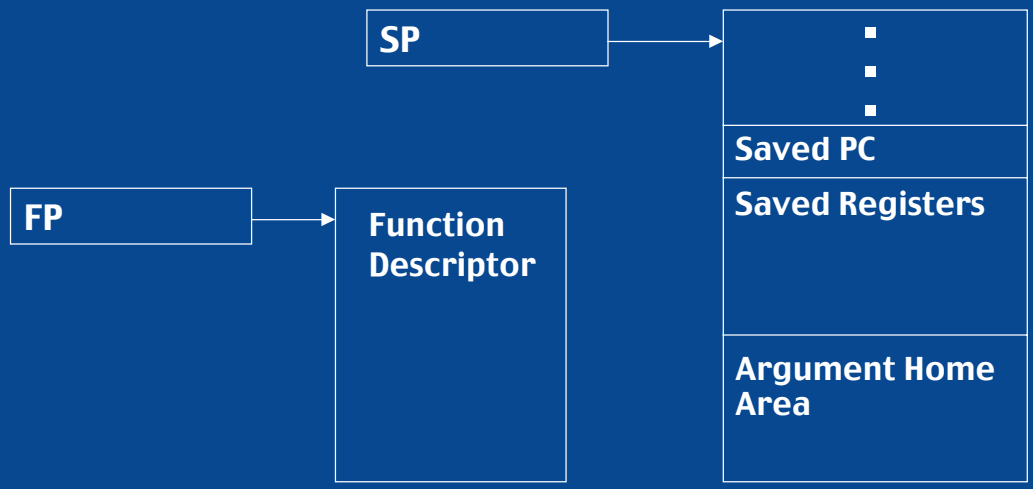
The Past

Alpha – Call Frame and Function Descriptor



The Past

Alpha – Call Frame and Function Descriptor



The Present

- Intel® defined Itanium® Calling Conventions
- Comparable to Alpha calling standard, but lacking
  - Argument count / information
  - VAX/Alpha floating point datatypes
  - Support for translated images
  - Definition of invocation context handle

Our Approach

- Adopt the mainstream Itanium® standards
  - Intel® calling standard
  - ELF object / image file format
  - DWARF debug information format
- Extend / specialize as needed to support existing VMS features

Benefits

- Intel® has accepted our extensions
- Intel compilers (C / C++, Fortran)
- Industry standard tools
  - Object file post-processors / analyzers
  - Linux linker

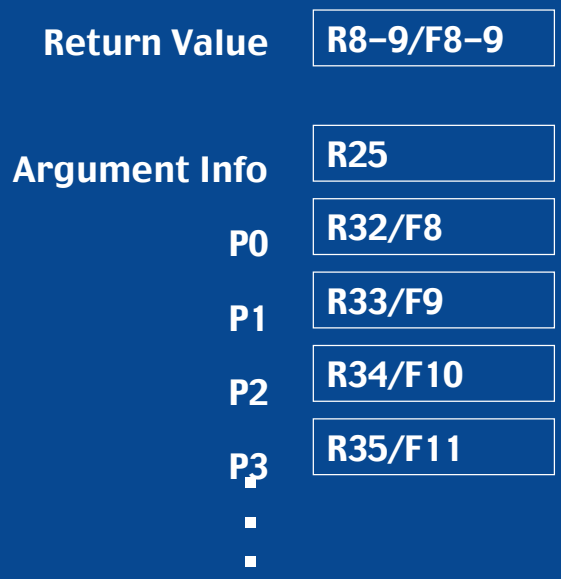
The Future

- OpenVMS Itanium® Calling Standard
- Based on industry standard Itanium Calling Conventions
- Extended for OpenVMS
  - Argument count / information register
  - VAX/Alpha floating point formats
  - Translated image support
  - Additional definitions

Differences Between Alpha and Itanium® Architectures

- Different registers for arguments and return value
- Rotating registers and separate register stack
- GP (global data pointer) register
- Different sets of scratch and saved registers
- PC range based condition handling

Argument List & Return Value



Floating Point Values

IEEE single & double precision (S & T float)

- Passed in floating point registers

VAX legacy floating point (F, D, G float)

- Passed in general registers

IEEE quad precision floating point

- Passed by reference (unlike Intel®)

Floating Point Library Interfaces

- Default floating point format in the Itanium® architecture is IEEE
- Existing APIs retain existing floating point format
- New APIs added as needed for IEEE format
- Language built-in math functions call APIs matching data type generated by the compiler
- Explicit floating point calls require action by the developer:
  - Compile with F / G float, or
  - Recode to use IEEE format API

Argument Information Register

- Argument data type
  - Integer (or address, etc.) in GR
  - Floating point in GR (F / D / G)
  - Floating point in FR (IEEE)

Extended Return Value

- Passed by reference using first parameter, as was done in OpenVMS VAX and Alpha
- R10/R11 are not used for extended return value
- Compatible with existing Alpha usage (e.g., porting mixed Fortran / assembly language or C)
- Not compatible with Intel® standard usage
- Applies only to
  - 128 bit floating point & complex
  - Floating point arrays
  - Other aggregates

Execution Environment

- R1 – Global Data Pointer (GP)
- R12 – Stack Pointer (SP)
- B0 – Function Return Address
- AR.PFS – Previous Function State
- AR.BSP – Backing Store Pointer
- Top of stack +16: Function Arguments P8 and up

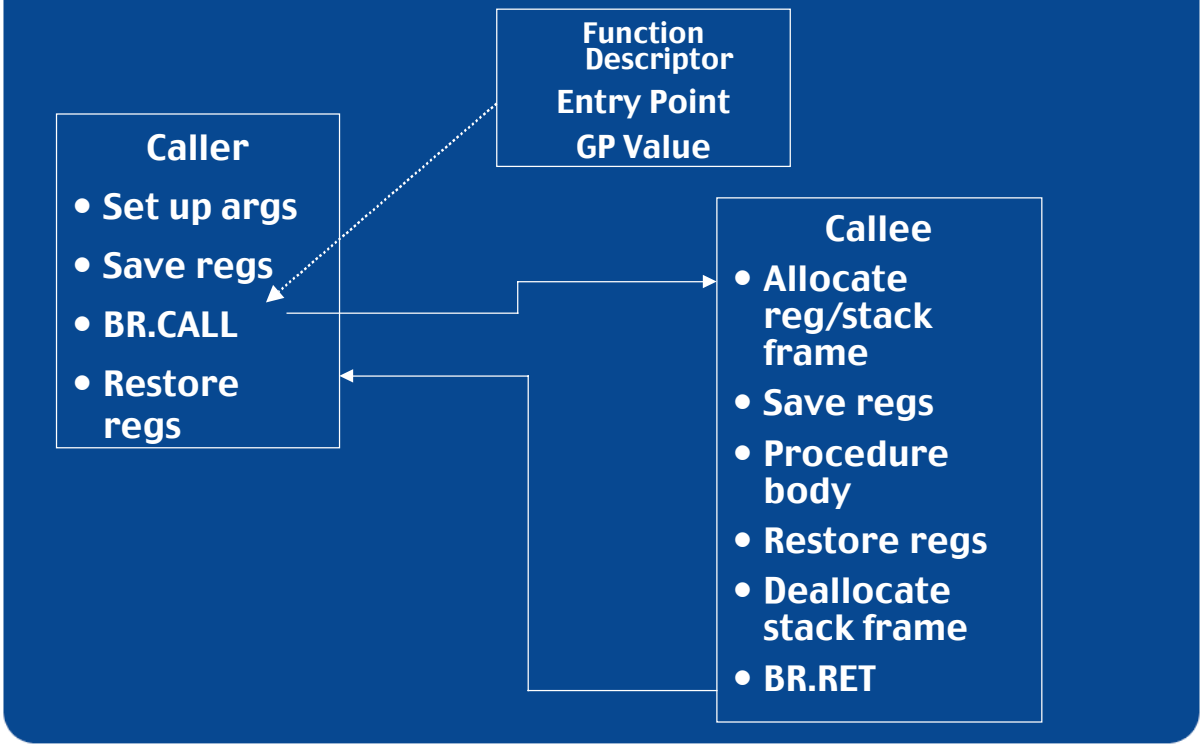
Register Classes

- Constant – constant value, may not be written
- Special – specific use defined by the Itanium® architecture or calling standard
- Automatic – managed by register stack engine; available to current procedure
- Preserved – (must be) preserved across procedure calls
- Scratch – not preserved across procedure calls; may be used between procedures by mutual agreement
- Volatile – not preserved during procedure calls; may not be used to pass information between procedures

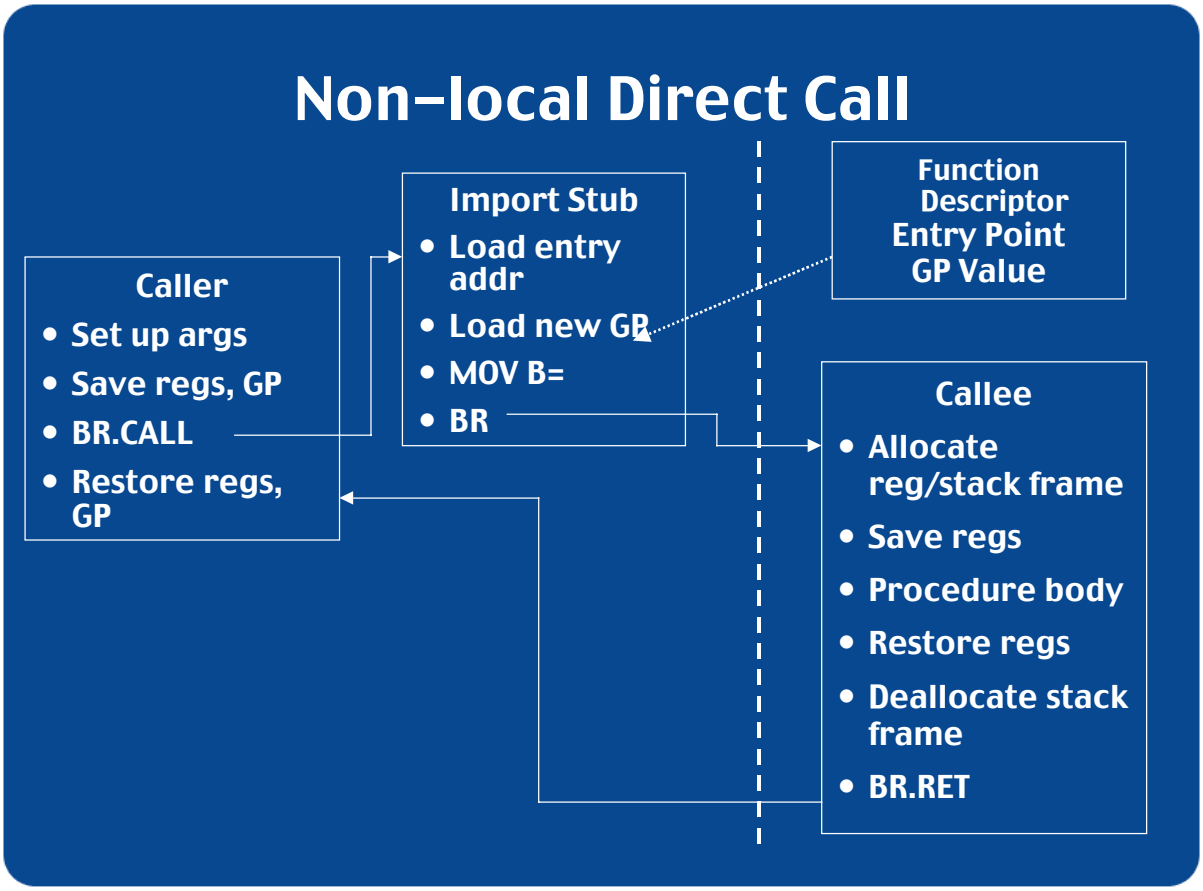
Procedure Call Linkage

- GP must be established before called procedure can execute
- All procedures in an executable (or shareable) image typically use a common GP
- Calling sequence depends on type of call
  - Local direct call
  - Non–local direct call
  - Indirect call

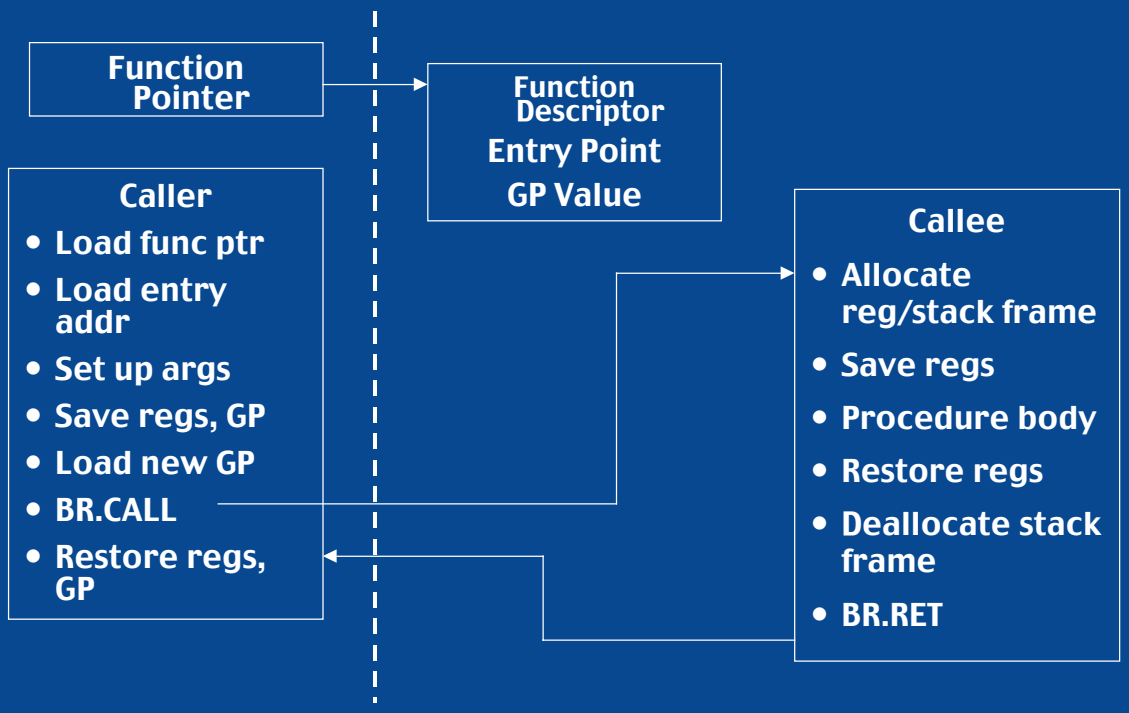
Local Direct Call



Non–local Direct Call



Indirect Call



Condition Handling

PC range based condition handling: there is no FP

- PC serves as key into a table of unwind descriptors
- Each unwind descriptor describes state of registers and stack for a range of code
- Invocation context block describes state of any active procedure
- Invocation context handle uniquely identifies an active procedure
  - Stack pointer if procedure has a memory stack
  - AR.BSP value otherwise

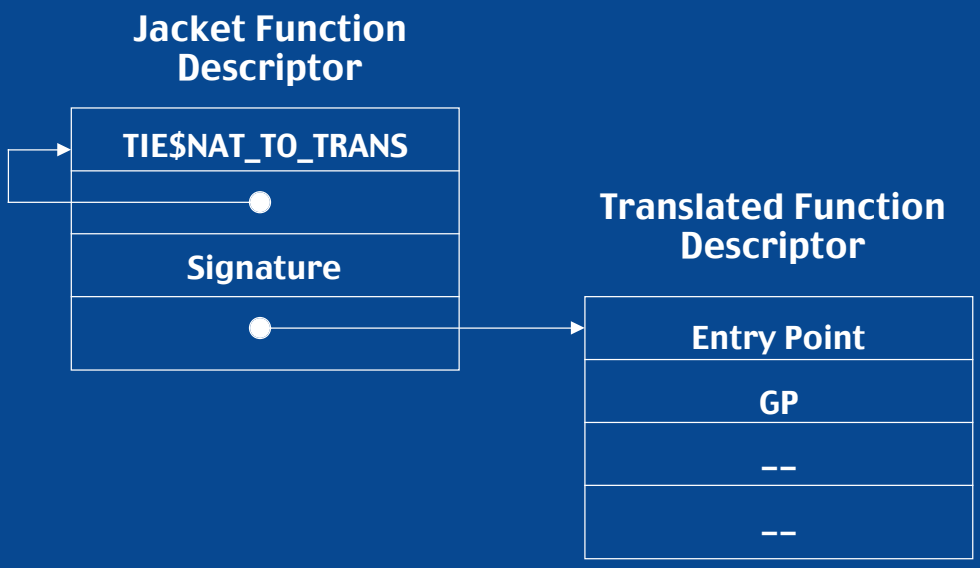
Invocation Context Services

- LIB$IPF_GET_CURR_INVO_CONTEXT – get invocation context for current procedure
- LIB$IPF_GET_INVO_HANDLE – get handle for invocation context
- LIB$IPF_GET_PREV_INVO_HANDLE – get handle for previous invocation context
- LIB$IPF_GET_PREV_INVO_CONTEXT – get invocation context for previous procedure
- LIB$IPF_GET_INVO_CONTEXT – get invocation context for specified procedure
- LIB$IPF_PUT_INVO_REGISTERS – modify registers of specified procedure invocation
- SYS$UNWIND – remove call frames from the stack

Translated Image Support

- Translated image = native code that does exactly what the original code did
- All data is the same as it was, including the stack
  - Translated Alpha code uses the Alpha calling standard
  - Translated VAX code uses the VAX calling standard
- Arguments must be transformed when calling between native and translated
- Stack must be interpreted to deliver exceptions to translated code

Translated Function Descriptor



Compiler Support for Migration

- Register Mapping
  - VAX Macro compiler maps Alpha register numbers to their Itanium® architecture equivalents
  - Bliss compiler supports user switchable mapping

Impact on Applications

- Mostly none
- Except for
  - Explicit register use
  - Knowledge of stack and exception frame format

More Information

Intel® IA-64 Architecture Software Developer's Manual

OpenVMS Calling Standard (V8.1 Doc Set)
- http://h71000.www7.hp.com/doc/v81final/pdf/OVMS_v81_call_std.PDF

http://www.hp.com/products1/evolution/alpha_retaintrust/openvms/

John.Covert@hp.com


invent