NAME

libunwind-ia64 -- IA-64-specific support in libunwind

INTRODUCTION

The IA-64 version of libunwind uses a platform-string of ia64 and, at least in theory, should be able to support all operating systems adhering to the processor-specific ABI defined for the Itanium Processor Family. This includes both little-endian Linux and big-endian HP-UX. Furthermore, to make it possible for a single library to unwind both 32- and 64-bit targets, the type unw_word_t is always defined to be 64 bits wide (independent of the natural word-size of the host). Having said that, the current implementation has been tested only with IA-64 Linux.

When targeting IA-64, the libunwind header file defines the macro UNW_TARGET_IA64 as 1 and the macro UNW_TARGET as ‘‘ia64’’ (without the quotation marks). The former makes it possible for platform-dependent unwind code to use conditional-compilation to select an appropriate implementation. The latter is useful for stringification purposes and to construct target-platform-specific symbols.

One special feature of IA-64 is the use of NaT bits to support speculative execution. Often, NaT bits are thought of as the ‘‘65-th bit’’ of a general register. However, to make everything fit into 64-bit wide unw_word_t values, libunwind treats the NaT-bits like separate boolean registers, whose 64-bit value is either TRUE (non-zero) or FALSE (zero).

MACHINE-STATE

The machine-state (set of registers) that is accessible through libunwind depends on the type of stack frame that a cursor points to. For normal frames, all ‘‘preserved’’ (callee-saved) registers are accessible. For signal-trampoline frames, all registers (including ‘‘scratch’’ (caller-saved) registers) are accessible. Most applications do not have to worry a-priori about which registers are accessible when. In case of doubt, it is always safe to *try* to access a register (via unw_get_reg() or unw_get_fpreg()) and if the register isn’t accessible, the call will fail with a return-value of -UNW_EBADREG.

As a special exception to the above general rule, scratch registers r15-r18 are always accessible, even in normal frames. This makes it possible to pass arguments, e.g., to exception handlers.

For a detailed description of the IA-64 register usage convention, please see the ‘‘Itanium Software Conventions and Runtime Architecture Guide’’, available at:

http://www.intel.com/design/itanium/downloads/245358.htm

REGISTER NAMES

The IA-64-version of libunwind defines three kinds of register name macros: frame-register macros,

normal register macros, and convenience macros. Below, we describe each kind in turn:

FRAME-REGISTER MACROS

Frame-registers are special (pseudo) registers because they always have a valid value, even though sometimes they do not get saved explicitly (e.g., if a memory stack frame is 16 bytes in size, the previous stack-pointer value can be calculated simply as sp+16, so there is no need to save the stack-pointer explicitly). Moreover, the set of frame register values uniquely identifies a stack frame. The IA-64 architecture defines two stacks (a memory and a register stack). Including the instruction-pointer (IP), this means there are three frame registers:

UNW_IA64_IP:

Contains the instruction pointer (IP, or ‘‘program counter’’) of the current stack frame. Given this value, the remaining machine-state corresponds to the register-values that were present in the CPU when it was just about to execute the instruction pointed to by UNW_IA64_IP. Bits 0 and 1 of this frame-register encode the slot number of the instruction. **Note:** Due to the way the call instruction works on IA-64, the slot number is usually zero, but can be non-zero, e.g., in the stack-frame of a signal-handler trampoline.

UNW_IA64_SP:

Contains the (memory) stack-pointer value (SP).

UNW_IA64_BSP:

Contains the register backing-store pointer (BSP). **Note:** the value in this register is equal to the contents of register ar.bsp at the time the instruction at UNW_IA64_IP was about to begin execution.

NORMAL REGISTER MACROS

The following normal register name macros are available:

UNW_IA64_GR:

The base-index for general (integer) registers. Add an index in the range from 0..127 to get a particular general register. For example, to access r4, the index UNW_IA64_GR+4 should be used. Registers r0 and r1 (gp) are read-only, and any attempt to write them will result in an error (-UNW_EREADONLYREG). Even though r1 is read-only, libunwind will automatically adjust its value if the instruction-pointer (UNW_IA64_IP) is modified. For example, if UNW_IA64_IP is set to a value inside a function func(), then reading UNW_IA64_GR+1 will return the global-pointer value for this function.

UNW_IA64_NAT:

The base-index for the NaT bits of the general (integer) registers. A non-zero value in these

registers corresponds to a set NaT-bit. Add an index in the range from 0..127 to get a particular NaT-bit register. For example, to access the NaT bit of r4, the index UNW_IA64_NAT+4 should be used.

UNW_IA64_FR:
The base-index for floating-point registers. Add an index in the range from 0..127 to get a particular floating-point register. For example, to access f2, the index UNW_IA64_FR+2 should be used. Registers f0 and f1 are read-only, and any attempt to write to indices UNW_IA64_FR+0 or UNW_IA64_FR+1 will result in an error (-UNW_EREADONLYREG).

UNW_IA64_AR:
The base-index for application registers. Add an index in the range from 0..127 to get a particular application register. For example, to access ar40, the index UNW_IA64_AR+40 should be used. The IA-64 architecture defines several application registers as ‘‘reserved for future use’’. Attempting to access such registers results in an error (-UNW_EBADREG).

UNW_IA64_BR:
The base-index for branch registers. Add an index in the range from 0..7 to get a particular branch register. For example, to access b6, the index UNW_IA64_BR+6 should be used.

UNW_IA64_PR:
Contains the set of predicate registers. This 64-bit wide register contains registers p0 through p63 in the ‘‘broad-side’’ format. Just like with the ‘‘move predicates’’ instruction, the registers are mapped as if CFM.rrb.pr were set to 0. Thus, in general the value of predicate register pN with N>=16 can be found in bit 16 + ((N-16)+CFM.rrb.pr) % 48.

UNW_IA64_CFM:
Contains the current-frame-mask register.

CONVENIENCE MACROS

Convenience macros are simply aliases for certain frequently used registers:

UNW_IA64_GP:
Alias for UNW_IA64_GR+1, the global-pointer register.

UNW_IA64_TP:
Alias for UNW_IA64_GR+13, the thread-pointer register.

UNW_IA64_AR_RSC:
Alias for UNW_IA64_GR+16, the register-stack configuration register.

UNW_IA64_AR_BSP:
Alias for UNW_IA64_GR+17. This register index accesses the value of register ar.bsp as of the time it was last saved explicitly. This is rarely what you want. Normally, you'll want to use UNW_IA64_BSP instead.

UNW_IA64_AR_BSPSTORE:
Alias for UNW_IA64_GR+18, the register-backing store write pointer.

UNW_IA64_AR_RNAT:
Alias for UNW_IA64_GR+19, the register-backing store NaT-collection register.

UNW_IA64_AR_CCV:
Alias for UNW_IA64_GR+32, the compare-and-swap value register.

UNW_IA64_AR_CSD:
Alias for UNW_IA64_GR+25, the compare-and-swap-data register (used by 16-byte atomic operations).

UNW_IA64_AR_UNAT:
Alias for UNW_IA64_GR+36, the user NaT-collection register.

UNW_IA64_AR_FPSR:
Alias for UNW_IA64_GR+40, the floating-point status (and control) register.

UNW_IA64_AR_PFS:
Alias for UNW_IA64_GR+64, the previous frame-state register.

UNW_IA64_AR_LC:
Alias for UNW_IA64_GR+65 the loop-count register.

UNW_IA64_AR_EC:
Alias for UNW_IA64_GR+66, the epilogue-count register.

THE UNWIND-CONTEXT TYPE

On IA-64, unw_context_t is simply an alias for ucontext_t (as defined by the Single UNIX Spec). This implies that it is possible to initialize a value of this type not just with unw_getcontext(), but also with getcontext(), for example. However, since this is an IA-64-specific extension to libunwind, portable code should not rely on this equivalence.

SEE ALSO

libunwind(3)

AUTHOR

David Mosberger-Tang
Email: **dmosberger@gmail.com**
WWW: **http://www.nongnu.org/libunwind/**.