Documentation Update for APAR PH61527

Preface

This document describes the IBM publication changes made to z/OS Language Environment by APAR PH61527.

The text marked in yellow are newly updated sections.

Language Environment vendor interfaces for AMODE 31 / AMODE 24 applications >
CALL linkage conventions >
Standard CALL linkage conventions >

…

Argument list format >
Argument passing - C linkage

…

When using C conventions, floating point parameters and structure return values are placed in storage whose address is passed as the first parameter, ~~vector data type value is returned in VR24~~vector data type values and 128-bit integer values are returned in VR24, other types are returned in GPR15. The +0 Entry Point prolog must relocate the return value into register 15 or in some cases into storage provided by the caller. The physical argument list in storage has space for the arguments which are passed in registers. The logical argument list consists of the physical argument list plus the contents of those registers used to pass arguments. ~~Vector arguments are loaded into VRs. Up to eight vector type value arguments are passed in VR24-31.~~ Except for arguments in the variable part of a vararg parameter list, up to a total of eight vector and 128-bit integer arguments are passed in VR24-31, and not passed in the argument area. For calls to unprototyped functions, where the caller cannot know if the called function contains a variable (vararg) portion, the argument list must be constructed to allow a call to either a vararg or non-vararg function. In this situation vector or 128-bit integer arguments passed in VRs are also passed in the argument list.

All addresses in the argument list are of a consistent width of 4 bytes. Each parameter takes up a multiple of 4 bytes.

…

- real or complex floating point numbers are fullword-aligned and may occupy one or more 4-byte slots in the argument list.
- ~~a vector argument is~~ vector and 128-bit integer arguments are full-word-aligned and occupy four 4-byte slots in the argument list.
- structures begin in the high order byte of a fullword and occupy an integral number of fullwords. Any padding bytes on the right end of the last full word are unused and their value is undefined.

FASTLINK CALL linkage conventions >

…

Argument list format

FASTLINK utilizes a logical argument list. Upon entry to the FASTLINK entry point at +16, the argument list is located in the argument area which is at a fixed location in what will be the callee's stack frame. At the +16 Entry Point, some of the argument values are passed in registers and some in storage. The physical argument list in storage has space for the arguments which are passed in registers. The logical argument list contains all of the arguments. The logical argument list consists of the physical argument list plus the contents of those registers used to pass arguments. Depending upon the type of the parameters, some arguments are loaded into the GPRs or the ~~FPRs ,~~ FPRs, or the VRs.

Argument passing

The logical argument list used with FASTLINK linkage is of the same format as the C linkage argument list, however, GPR1 does not point to the argument list. Instead, the arguments are placed into the argument area of the callee's stack frame or certain general purpose or floating point ~~registers ,~~ registers, or vector registers.

In FASTLINK, the first three words of the virtual argument list are loaded into GPR1-3 if they represent indirect arguments or direct value arguments of data types other than floating point (real or complex) ~~or vector~~ , vector, or 128-bit integer. If a direct value floating point argument (real or complex) begins in the first 3 argument words, it is loaded into an appropriate number of floating point registers FPR0 through FPR6. Only one such floating point value is loaded into a floating point register. If a second floating point value begins in the first three virtual argument words, it is located in storage. ~~Up to eight vector arguments are passed directly in VR24-31 and VR24 is used for returns as well. When a floating point or vector argument is loaded in FPRs or VRs, the contents of the GPRs corresponding to those argument words are unpredictable and are not preserved over the call.~~ Up to a total of eight vector and 128-bit integer arguments are passed directly in VR24-31 and VR24 is used for returns as well. When floating point, vector or 128-bit integer arguments are loaded in FPRs or VRs, the contents of the GPRs corresponding to those argument words are unpredictable and are not preserved over the call.

Register conventions

…

Register	Description			
GPR0	Undefined, Not preserved.			
GPR1-3	First argument words, or undefined: 1. If no arguments exist 2. If the corresponding arguments are floating point scalars or floating point scalar complex values. 3. If the corresponding arguments are vector ~~values~~ or 128-bit integer values.			
GPR1-3	When are GPR Registers 1-3 preserved?			
GPR1-3	Logical Argument Word 1	Logical Argument Word 2	Logical Argument Word 3	Registers Preserved
GPR1-3	empty	empty	empty	GPR1-3
GPR1-3	argument	empty	empty	GPR2-3
GPR1-3	argument	argument	empty	GPR3
GPR1-3	argument	argument	argument	none

Note: When specified, empty means that there is no corresponding parameter value. Thus, a call with no parameters preserves the GPRs 1-3. ~~A call with one floating point extended parameter, or vector parameter uses the FPRs or VRs to contain the floating/vector value and, except for a very special case, GPRs 1-3 have an undefined value and are not preserved over the call.~~

A call with one floating point extended parameter, or vector parameter, or 128-bit integer parameter uses the FPRs or VRs to contain the parameter value and, except for a very special case, GPRs 1-3 have an undefined value and are not preserved over the call.

VR16-23	Undefined. Preserved
VR24-31	Vector or 128-bit integer type parameters or undefined. VR24 is used for returns as well. They are not preserved.
ARs	Undefined Preserved.
Condition register	Undefined. Not preserved.
Program mask	As documented in this information.

Extra Performance Linkage (XPLINK) CALL linkage conventions >

...

Stack frame mapping >

Argument list format >
Argument passing register conventions

The following tables describe the XPLINK register conventions used for passing arguments.
...

Register	Conventions on function entry Exit	Volatility
VR 0-7	undefined	not preserved
VR 8-15	undefined	Bytes 0-7 are preserved due to overlap with FPR8-15, bytes 8-15 are not preserved.
VR 16-23	undefined	preserved
VR 24-31	Vector or 128-bit integer type parameters or undefined. VR 24 is used for returns.	not preserved

Stack frame mapping >
Argument list format >
Argument passing
...
Since support of stack extensions may require copying of argument lists to different storage locations, the argument list must not include arguments that are pointers to locations in the argument list. The rules for argument passing in registers are as follows:
- The first 3 (4-byte) words of the argument area, regardless of their composition or source, are passed in GPRs 1, 2, and 3, and not in the argument area (although space for these words is reserved in the argument area), except for vector values, 128-bit integer values and floating point values, including the real or imaginary constituents of complex types.
Not every language supports complex types. For the purposes of argument passing and function return values, in every language, every aggregate that is (a) not a union, and (b) contains exactly two floating-point types of the same size (~~4,8,~~4, 8, or 16 bytes) is treated as a complex type.

...
- Except for arguments in the variable part of a vararg parameter list, up to a total of eight vector and 128-bit integer arguments are passed in VR24-31, and not passed in the argument area, although space is set aside for these arguments in the argument area. If a vector argument or a 128-bit integer argument occupies one of the first three words in the argument area, a prototype for the function is visible, and the argument is not part of the vararg portion of a parameter list, the corresponding GPR's value is undefined on entry to the called function.

- Normally, arguments passed in registers are not stored in the argument list although a slot in the argument list is reserved for them.

There is an exception to this rule: if it is required that part of a floating point or vector or 128-bit integer value be stored in the argument area, then the entire floating or vector or 128-bit integer value is stored in the argument area. This situation also arises in calls to unprototyped functions or in the vararg portion of a parameter list when part of the floating point ~~or vector~~, vector or 128-bit integer parameter is in the first three words of the argument area. ~~For more information, see examples f13, f18, and f20 in CALL linkage argument examples.~~
For calls to unprototyped functions, where the caller cannot know if the called function contains a variable (vararg) portion, the argument list must be constructed to allow a call to either a vararg or non-vararg function. ~~In this situation: floating-point and vector arguments in the first 3 words of the parameter list are passed in GPRs, FPRs and VRs; other floating point or vector arguments passed in FPRs or VRs are also passed in the argument list.~~ In this situation:
- arguments (regardless of their types) in the first 3 words of the parameter list are passed in GPRs
- arguments (regardless of their types) beyond the first 3 words of the parameter list are stored onto argument area
- floating point arguments are also passed in FPRs, in addition to being passed in GPRs or argument area
- vector or 128-bit integer arguments are also passed in VRs, in addition to being passed in GPRs or argument area

For more information, see examples f13, f18, f20, f28, f29, f31, f32, f33, and f35 in CALL linkage argument examples.

To support varargs functions, calls to unprototyped functions, and compatibility with older linkages, the minimum argument area length must be 16 bytes. This allows the compiler to map the first three arguments in storage as well as registers and provides for compatibility with linkages that have a hidden last parameter.
Stack frame mapping >
Argument list format >
Argument passing >
Call Descriptor - Parameter Descriptions
…
It is the compiler's responsibility to pass the maximum number of parameters that fit this encoding scheme so that the parameters in registers will match between caller and called function. When calling a vararg routine, no argument in the variable portion of the argument is passed in a Floating Point Register or Vector Register. When calling unprototyped functions floating point, ~~or~~vector or 128-bit integer parameters are passed in FPRs or VRs matching this encoding scheme and are also shadowed, by the caller, in GPRs or memory. Call descriptors are not required for calls to unprototyped functions whose return value is not examined by the caller.

Stack frame mapping >
Argument list format >
Function Return Values
Functions return their values according to type:

1. Integral and pointer data types that are less than or equal to 32 (≤ 32) bits in length are widened to 32 bits and returned in GPR3.

2. Integral data types greater than 32 bits and less than or equal to 64 (≤ 64) bits in length are widened to 64 bits and returned in GPR2 (the leftmost 32 bits) and GPR3 (the rightmost).

3. Integral data types greater than 64 bits and less than or equal to 128 (≤ 128) bits in length are widened to 128 bits and returned in VR24.

4. Floating point types, including complex types, are returned FPR0, FPR2, FPR4 and FPR6, using as many registers as required.

Language Environment vendor interfaces for AMODE 64 applications > CALL linkage conventions for AMODE 64 applications > XPLINK CALL linkage conventions for AMODE 64 applications >

...

Stack Format >
Function calls >
Argument passing register conventions

The following tables describe the XPLINK register conventions used for passing arguments.
...

Register	Conventions on function entry and exit	Volatility
VR 0-7	undefined	not preserved
VR 8-15	undefined	Bytes 0-7 are preserved due to overlap with FPR8-15, bytes 8-15 are not preserved
VR 16-23	undefined	preserved
VR 24-31	Entry: Vector or 128-bit integer type parameters or undefined.	not preserved
VR 24-31	Exit: VR24 is used for returns.	not preserved

Stack Format >
Function calls >
Argument passing

…

The rules for argument passing in registers are as follows:

- The first three doublewords of the argument area, regardless of their composition or source, are passed in GPRs 1, 2, and 3, and not in the argument area, except for:

    - Floating point values, including the real or imaginary constituents of complex types

    - Vector arguments

    - 128-bit integer arguments

…

- Except for arguments in the variable part of a vararg parameter list, up to a total of eight vector ~~arguments~~ and 128-bit integer arguments are passed in VR24-31, and not passed in the argument area, although space is set aside for these arguments in the argument area. If a vector argument or 128-bit integer argument occupies one of the first three doublewords ~~words~~ in the argument area, a prototype for the function is visible, and the argument is not part of the vararg portion of a parameter list, the corresponding GPR's value is undefined on entry to the called function.

- Normally, arguments passed in registers are not stored in the argument list although a doubleword in the argument list is reserved for them.

When calling a vararg routine, no argument in the variable portion of the argument is passed in a Floating Point Register or Vector Register.

For calls to unprototyped functions, where the caller cannot know whether the called function contains a variable vararg portion, the argument list must be constructed to allow a call to either a vararg or non-vararg function. ~~In this situation, floating-point or vector arguments in the first three doublewords of the parameter list are passed in GPRs, FPRs or VRs. Other floating point arguments passed in FPRs or VRs are also passed in the argument list.~~ In this situation:
- arguments (regardless of their types) in the first 3 doublewords of the parameter list are passed in GPRs
- arguments (regardless of their types) beyond the first 3 doublewords of the parameter list are stored onto argument area
- floating point arguments are also passed in FPRs, in addition to being passed in GPRs or argument area

- vector or 128-bit integer arguments are also passed in VRs, in addition to being passed in GPRs or argument area

To support vararg functions and calls to unprototyped functions, the minimum argument area length must be 32 bytes.

~~The compiler passes the maximum number of parameters that fit this encoding scheme so the parameters in registers match between caller and called functions. When calling a vararg routine, no argument in the variable portion of the argument is passed in a Floating Point Register or Vector Register. When calling unprototyped functions, floating point or vector parameters are passed in FPRs or VRs matching this encoding scheme and are also shadowed by the caller, in GPRs or memory.~~

Stack Format >
Function calls >
Function return values
Functions return their values according to type:

1. Integral and pointer data types ≤64 bits in length are widened to 64 bits and returned in GPR3.

2. Integral data types > 64 bits and ≤ 128 bits in length are widened to 128 bits and returned in VR24.

3. Floating point types, including complex types, are returned FPR0, FPR2, FPR4 and FPR6, using as many registers as required.
Restriction: Not every language supports complex types. For the purposes of argument passing and function return values, in every language every aggregate that is (a) not a union, and (b) contains exactly two floating-point types of the same size (4, 8, or 16 bytes) is treated as a complex type.

…

CALL linkage argument examples >

XPLINK CALL linkage argument examples >

"Parameter Adjust" is not used for AMODE 64 applications only.
The following example shows "by reference" parameters. In this example, "Parameter Adjust" is always zero and arguments are never passed in floating point registers. The value of the high-order bit on the last, or any, reference parameter is not defined here; this is left to the implementation, possibly specified by language constructs such as #pragma in C.
…

Prototype:	f31(	int,	int,	vector double)			
Offset in argument list		+0	+4	+8	+12	+16	+20
Stored in argument list		No	No	No	No	No	Yes
Passed in Registers		GPR1	GPR2	VR24			
Parameter Adjust		(none)					

The following figures show how 128-bit integer type arguments are passed. A 128-bit integer argument is double-word-aligned and occupy 16 bytes in the argument list. And in unprototyped calls, linkage need to match the conventions expected by both vararg and non-vararg functions.

P e:	f32(	ec d ble,	igned __in 128,	in)
Offset in argument list		+0	+16	+32
Stored in argument list		No	No	Yes
Passed in Registers		VR24	VR25	
Parameter Adjust		(none)		

P e:	f33(	in ,	igned __in 128,	in)
Off e in a g men li		+0	+4	+20
S ed in a g men li		No	No	Yes
Pa ed in Regi e		GPR1	VR24	
Pa ame e Adj		(none)		

Prototype:		(none)					
Actual Parameters		int	int	unsigned __int128			
Offset in argument list		+0	+4	+8	+12	+16	+20
Stored in argument list		No	No	Yes	Yes	Yes	Yes
Passed in Registers		GPR1	GPR2	GPR3			
				VR24			
Parameter Adjust		(none)					

Prototype:	f34(	int,	...)				
Actual Parameters		int	int	unsigned __int128			
Offset in argument list		+0	+4	+8	+12	+16	+20
Stored in argument list		No	No	Yes	Yes	Yes	Yes
Passed in Registers		GPR1	GPR2	GPR3			
Parameter Adjust		(none)					

	+0	+4	+8	+12	+16	+20	
	No	No	No	No	No	Yes	
	GPR1	GPR2	VR24				
	(none)						