x64 prolog and epilog

Article • 08/03/2021

Every function that allocates stack space, calls other functions, saves nonvolatile registers, or uses exception handling must have a prolog whose address limits are described in the unwind data associated with the respective function table entry. For more information, see x64 exception handling. The prolog saves argument registers in their home addresses if necessary, pushes nonvolatile registers on the stack, allocates the fixed part of the stack for locals and temporaries, and optionally establishes a frame pointer. The associated unwind data must describe the action of the prolog and must provide the information necessary to undo the effect of the prolog code.

If the fixed allocation in the stack is more than one page (that is, greater than 4096 bytes), then it's possible that the stack allocation could span more than one virtual memory page and, therefore, the allocation must be checked before it's allocated. A special routine that's callable from the prolog and which doesn't destroy any of the argument registers is provided for this purpose.

The preferred method for saving nonvolatile registers is to move them onto the stack before the fixed stack allocation. If the fixed stack allocation is performed before the nonvolatile registers are saved, then most probably a 32-bit displacement is required to address the saved register area. (Reportedly, pushes of registers are as fast as moves and should remain so for the foreseeable future in spite of the implied dependency between pushes.) Nonvolatile registers can be saved in any order. However, the first use of a nonvolatile register in the prolog must be to save it.

Prolog code

The code for a typical prolog might be:

MASM

```
mov    [RSP + 8], RCX
push   R15
push   R14
push   R13
sub    RSP, fixed-allocation-size
lea    R13, 128[RSP]
...
```

This prolog stores the argument register RCX in its home location, saves nonvolatile registers R13-R15, allocates the fixed part of the stack frame, and establishes a frame pointer that points 128 bytes into the fixed allocation area. Using an offset allows more of the fixed allocation area to be addressed with one-byte offsets.

If the fixed allocation size is greater than or equal to one page of memory, then a helper function must be called before modifying RSP. This helper, `__chkstk`, probes the to-be-allocated stack range to ensure that the stack is extended properly. In that case, the previous prolog example would instead be:

MASM

```
mov    [RSP + 8], RCX
push   R15
push   R14
push   R13
mov    RAX,  fixed-allocation-size
call   __chkstk
sub    RSP, RAX
```

```
    lea    R13, 128[RSP]
    ...
```

The `__chkstk` helper will not modify any registers other than R10, R11, and the condition codes. In particular, it will return RAX unchanged and leave all nonvolatile registers and argument-passing registers unmodified.

Epilog code

Epilog code exists at each exit to a function. Whereas there is normally only one prolog, there can be many epilogs. Epilog code trims the stack to its fixed allocation size (if necessary), deallocates the fixed stack allocation, restores nonvolatile registers by popping their saved values from the stack, and returns.

The epilog code must follow a strict set of rules for the unwind code to reliably unwind through exceptions and interrupts. These rules reduce the amount of unwind data required, because no extra data is needed to describe each epilog. Instead, the unwind code can determine that an epilog is being executed by scanning forward through a code stream to identify an epilog.

If no frame pointer is used in the function, then the epilog must first deallocate the fixed part of the stack, the nonvolatile registers are popped, and control is returned to the calling function. For example,

MASM

```
    add      RSP, fixed-allocation-size
    pop      R13
    pop      R14
    pop      R15
    ret
```

If a frame pointer is used in the function, then the stack must be trimmed to its fixed allocation prior to the execution of the epilog. This action is technically not part of the epilog. For example, the following epilog could be used to undo the prolog previously used:

MASM

```
    lea      RSP, -128[R13]
    ; epilogue proper starts here
    add      RSP, fixed-allocation-size
    pop      R13
    pop      R14
    pop      R15
    ret
```

In practice, when a frame pointer is used, there is no good reason to adjust RSP in two steps, so the following epilog would be used instead:

MASM

```
    lea      RSP, fixed-allocation-size - 128[R13]
    pop      R13
    pop      R14
    pop      R15
    ret
```

These forms are the only legal ones for an epilog. It must consist of either an `add RSP,constant` or `lea RSP,constant[FPReg]`, followed by a series of zero or more 8-byte register pops and a `return` or a `jmp`. (Only a subset of `jmp` statements are allowable in the epilog. The subset is exclusively the class of `jmp` statements with ModRM memory references where ModRM mod field value is 00. The use of `jmp` statements in the epilog with ModRM mod field value 01 or 10 is prohibited. See Table A-15 in the AMD x86-64 Architecture Programmer's Manual Volume 3: General Purpose and System Instructions, for more info on the allowable ModRM references.) No other code can appear. In particular, nothing can be scheduled within an epilog, including loading of a return value.

When a frame pointer is not used, the epilog must use `add RSP,constant` to deallocate the fixed part of the stack. It may not use `lea RSP,constant[RSP]` instead. This restriction exists so the unwind code has fewer patterns to recognize when searching for epilogs.

Following these rules allows the unwind code to determine that an epilog is currently being executed and to simulate execution of the remainder of the epilog to allow recreating the context of the calling function.

See also

x64 Software Conventions

Feedback