November 11, 2015

CODE SECURITY WITH VARIABLE-LENGTH INSTRUCTION ENCODING

Bruno De Backer

Ondrej Sykora

Attorney Docket No.: 16113-6802001

CODE SECURITY WITH VARIABLE-LENGTH INSTRUCTION ENCODING

TECHNICAL FIELD

[0001] This disclosure generally relates to instruction encoding.

BACKGROUND

[0002] Processors may use fixed-length instructions or variable-length instructions. Fixed-length instructions may have an advantage of simplicity, and security may be implemented by having instructions be aligned in memory, e.g., having all instructions start on odd addresses. However, code size may be larger, inducing pressure on a memory subsystem.

[0003] On the other hand, variable-length instructions may be more complex to decode, but code size may be reduced. However, because instructions may have different sizes, branching to any unprotected address in the code must be enabled. Because of this, security issues may arise since code that seems innocuous when run from address N may be decoded in a completely different way at address N + 1 and have a malicious behavior. Moreover, determining which of the possible interpretations of instructions will be executed without actually running a program is in general an undecidable problem.

SUMMARY

[0004] In general, an aspect of the subject matter described may involve a process for providing code security with variable-length instruction encoding. Code security may be provided through encoding instructions according to a set of rules. Such encoding may delimit the beginning of instructions using a specific bit, delimit the beginning of a basic block using another specific bit, enable handling of variable length constants that code size, and make code validation decidable and deterministic. Accordingly, code may be scanned once to verify that the code can be decoded into actual instructions, non-computed branches can be validated, and computed branches may trap if they branch to other places than the beginning of a basic starting block.

DESCRIPTION OF DRAWINGS

[0005] FIG. 1A is an illustration of a format of an example first byte of a variable-length instruction.

[0006] FIG. 1B is an illustration of a format of an example subsequent byte of a variable-length instruction.

[0007] FIG. 2 is a flowchart of an example process for processing bytes of a variable-length instruction.

DETAILED DESCRIPTION

[0008] FIG. 1A is an illustration of a format of an example first byte of an instruction. The format represents each of the eight bits of a byte in individual cells, where the most significant bit is on the left and the least significant bit is on the right.

[0009] The format specifies that that the value of the most significant bit of bytes, which may be referred to as “S” indicates whether the byte is the first byte of an instruction. Instructions that are variable-length may include instructions that are a single byte, e.g., non multi-byte instructions, or instructions that are multiple bytes, e.g., multi-byte instructions. Accordingly, the first byte of an instruction for a non multi-byte instruction may be the only byte of the instruction, and the first byte of an instruction for a multi-byte instruction may be the beginning byte of the instruction. In FIG. 1A, bit S is shown with a value of “1.” If the byte is the first byte of an instruction, the value of bit S is “1.” If the byte is not the first byte of an instruction, the value of bit S is “0.” Accordingly, a hardware exception may result when processing a byte whose bit S has a value “0” as a first byte of an instruction.

[0010] The format further specifies that the value of bit B of first bytes indicates whether the byte is the beginning of a basic block. Bytes that are the beginning of basic blocks may be the target of branches, and bytes that are not the beginning of basic must not be the target of branches. In FIG. 1A, the second most significant bit is shown as “B.” If the byte is a first byte of a basic block, the value of bit B is “1.” If the byte is not the first byte of a basic block, the value of bit B is “0.” Accordingly, a hardware exception may result when branching to a byte that has a bit B with a value of “0.”

[0011] The format further specifies that that the value of the third most significant bit of first bytes indicates whether the byte is the first byte of a multi-byte instruction. In FIG. 1A, the third most significant bit is shown as "M." If the byte is the first of a multi-byte instruction, the value of bit M is "1." If the byte is not a first byte of a multi-byte instruction, the value of bit M is "0."

[0012] The format further specifies that that the value of the fourth to eighth most significant bits of first bytes represent the payload of the instruction. In FIG. 1A, the remaining bits of the byte are shown as “P4,” “P3,” “P2,” “P1,” and “P0,” respectively. In cases where the first byte is a first byte of a non multi-byte instruction, the fourth to eighth most significant bits of first bytes may represent the action of the instruction or parameters for the instruction. For example, the payload may indicate an action of “immediate load,” the value to load, and the register to load the value in. In cases where the first byte is a first byte of a multi-byte instruction, the fourth to eighth most significant bits of first bytes may represent a number of subsequent bytes in the instruction. For example, the bits may have a value of four, representing that the instruction includes four more subsequent bytes. Depending on a word size of an underlying architecture, less than five bits may be used, and the remaining bits may be used to specify the instruction itself.

[0013] Accordingly, in an example, a byte representing bits “10000000” may indicate that the byte is the first byte of an instruction, the byte is not for the first instruction of a basic block, the byte represents an instruction that is not multi-byte, and the code of the instruction is “00000.” In another example, a byte representing bits “11000000” may indicate that the byte is a first byte of an instruction, the byte is for a first instruction of a basic block, the byte represents an instruction that is not multi-byte, and the code of the instruction is “00000.”

[0014] FIG. 1B is an illustration of a format of an example subsequent byte of a variable-length instruction. The format specifies that that the value of the most significant bit of bytes that are subsequent bytes of instructions, i.e., not first bytes of instructions, is “0.” The remaining bits of the byte contain the payload for the instruction.

[0015] An example multi-byte instruction following formats 100 and 150 may be “11101010 0xxxxxxb63 0b62-----b56 0b55-----b49 0b48-----b42 0b41-----b35 0b34-----b28 0b27-----b21 0b20-----b14 0b13-----b7 0b6-----b0,” where 0xxxxxxb63 contains the opcode of the load instruction, the register to which it applies, and the 63rd bit of the immediate value, 0b62-----b56 contains the 62nd bit through 56th bit of the immediate value, 0b55-----b49 contains the 55th bit through 49th bit of the immediate value, etc. In accordance with the formats 100 and 150, the bit S of the first byte is “1,” indicating that the first byte is a first byte of an instruction. The bit B of the first byte is “1,” indicating that the first byte is a beginning of a basic block. The bit M of the first byte is “1,” indicating that the first byte is for an instruction that is a multi-byte instruction. The fourth through eighth significant bits of the first byte have a value of ten, indicating that instruction includes ten bytes following the first byte.

[0016] Other formats other than those shown in FIGs. 1A and 1B may be used that have bits that explicitly mark the start of an instruction, start of a basic block/branch target, and whether the instruction is a multi-byte instruction. The precise position of these bits in the bytes in a format may not be significant. Having these bits at any other position may work equally well. The encoding described above is based on the convention set by e.g. UTF-8 (keep metadata in the most significant bits, and the payload in the lower bits), but other encoding conventions may be used (for example, keeping metadata in the least significant bits). In some implementations, different values of indicator bits may be used. For instance, instead of the most significant bits of first bytes of instructions having values of “1” and the most significant bits of subsequent bytes of instructions having values of “0,” the most significant bits of first bytes of instructions may have values of “0” and the most significant bits of subsequent bytes of instructions may have values of “1.”

[0017] FIG. 2 is a flowchart of an example process 200 for processing bytes of a variable-length instruction. The process 200 may include determining if the particular byte is a first byte of an instruction (210). For example, a processor may determine if bit

S of the particular byte has a value of “1.” If the processor determines that bit S of the particular byte does not have a value of “1,” the processor may generate a hardware exception and stop execution (212). If the processor determines that bit S of the particular byte does have a value of “1,” the processor may continue the process 200.

[0018] The process 200 may include determining if the particular byte is a target of a branch and a beginning of a basic block, or is not a target of a branch (220). If the byte is a target of a branch, the process 200 may determine whether bit B of the particular byte has a value of “1” (222). For instance, the processor may determine that the particular byte is being jumped to, and in response, determine that bit B of the particular byte has a value of “1.” In the example, the processor may then continue with process 200. In another example, the processor may determine that the particular byte is being jumped to, and in response, determine that bit B of the particular byte has a value of “0” and in response, generate a hardware exception and stop execution (224). In yet another example, the processor may determine that the particular byte is not being jumped to and in response, may then continue with process 200.

[0019] The process 200 may include determining if the instruction is a multi-byte instruction (230). For example, a processor may determine if bit M of the particular byte has a value of "0." If the processor determines that bit M of the particular byte has a value of "0," the processor may determine that the instruction is not a multi-byte instruction and continue with the process 200. If the processor determines that bit M of the particular byte does not have a value of "0," the processor may determine that the instruction is a multi-byte instruction and determine whether the bit S of the remaining bytes in the multi-byte all have a value of “0” (232). If they do, the process 200 may continue. If they do not, the process 200 may generate a hardware exception and stop execution (234).

[0020] The process 200 may include processing the instruction (240A and 240B). For example, where the processor determined that the instruction is not a multi-byte instruction, the processor may process the single byte, e.g., perform a NOP, (240A). In another example, where the process determined that the instruction is a multi-byte instruction, the processor may obtain the values of the remaining subsequent bytes of the multi-byte instructions, and then process the instruction (240B). After the instruction is processed, the process 200 may then repeat again for another instruction.

[0021] In some instruction sets, constants of different numbers of bits may be part of the instruction code. For instance, some 64-bit constants may be represented with fewer bits. Accordingly, an alternative encoding for an immediate load at the beginning of a building block could be 111LLLLL 0MMMMMMM 0aaaaaaa 0bbbbbbb 0ccccccc 0ddddddd 0eeeeeee 0fffffff …, where 0MMMMMMM represents the opcode chosen for that particular load immediate instruction. Here, the length of the constant may be deduced from the length of the instruction in LLLLL. Therefore, a load of a small constant, e.g., “1,” to a 64-bit register may be encoded as 11100010 0MMMMMMM 00000001, reducing the number of bytes used in representing the instruction.

[0022] In some implementations, the process may be used to encapsulate existing byte codes or instruction encodings. For example, encapsulating an instruction of *n* bytes may be performed with the following. For each instruction, the first byte may be 1B1LLLLL, where B is 1 when an instruction is the beginning of a basic block and LLLLL is a ceil value of the number of bytes in the instruction times eight divided by seven, e.g., ceil(sizeof(instruction) * 8 / 7). Each of the following bytes may have 0 as their most significant bit, and contain 7 bits of the instruction of the existing architecture or bytecode.

ABSTRACT OF THE DISCLOSURE

Methods, systems, and apparatus, including computer programs encoded on a computer storage medium, for providing code security with variable-length instruction encoding. An aspect may include encoding instructions according to a set of rules. Such encoding may delimit the beginning of instructions using a specific bit, delimit the beginning of a basic block using another specific bit, enable handling of variable length constants that code size, and make code validation decidable and deterministic. Accordingly, code may be scanned once to verify that the code can be decoded into actual instruction, non-computed branches can be validated, and computed branches may trap if they branch to other places than the beginning of a basic starting block.

Format for the first byte of a instruction

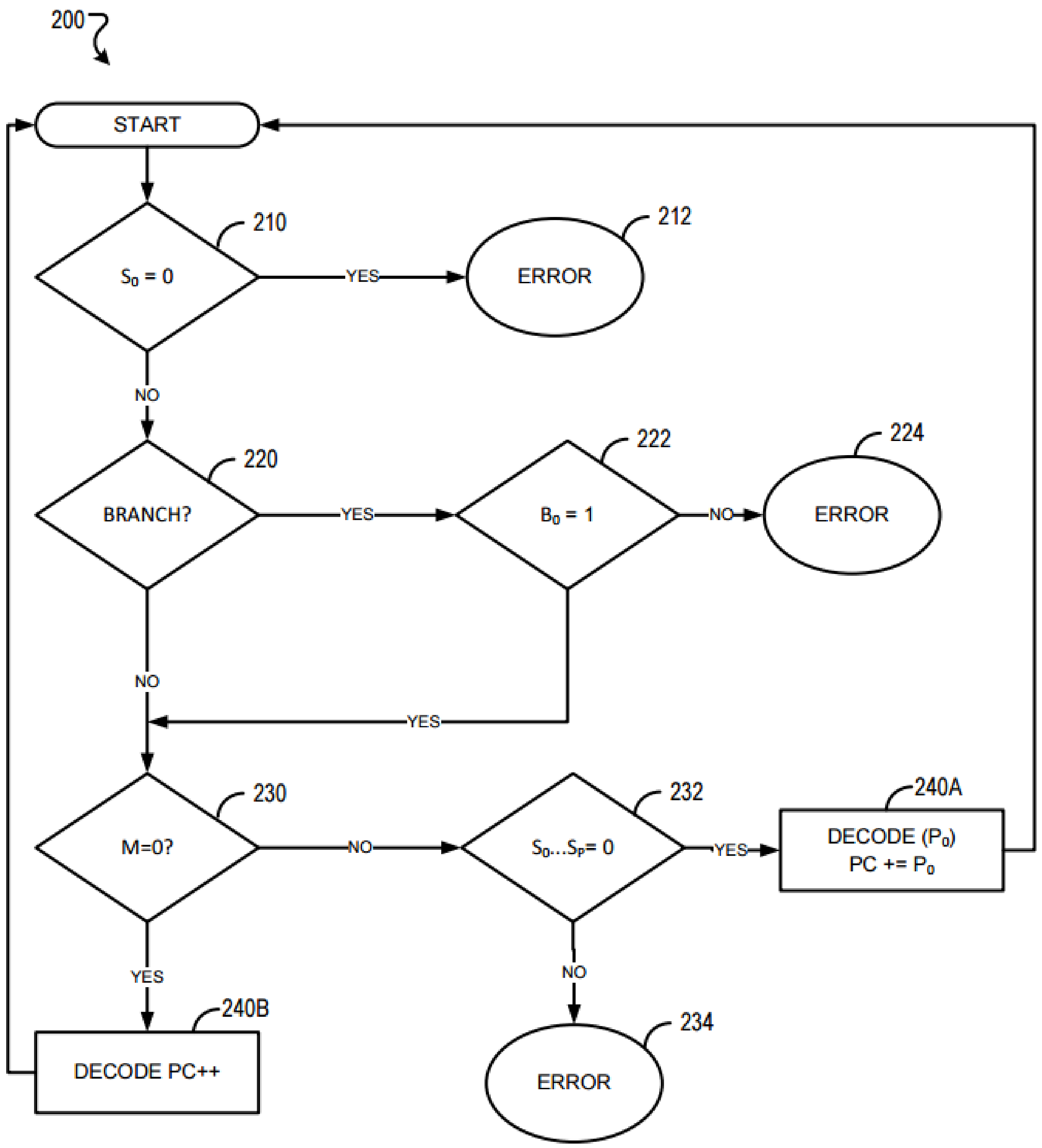
1	B	M	P4	P3	P2	P1	P0

FIG. 1A

Format for the subsequent bytes of an instruction

0	P6	P5	P4	P3	P2	P1	P0

FIG. 1B



Byte 0				Byte 1		Byte 2		Byte 3	
S_0	B_0	M_0	P_0	S_1	P_1	S_2	P_2	S_3	P_3

FIG. 2