Monthly Research

Intel Memory Protection Extensions Overview

F F R I , Inc
http://www.ffri.jp

Ver 1.00.02

Intel® Memory Protection Extensions(Intel MPX)

- Intel MPX is a set of instructions of x86
  - Hardware support for bounds checking

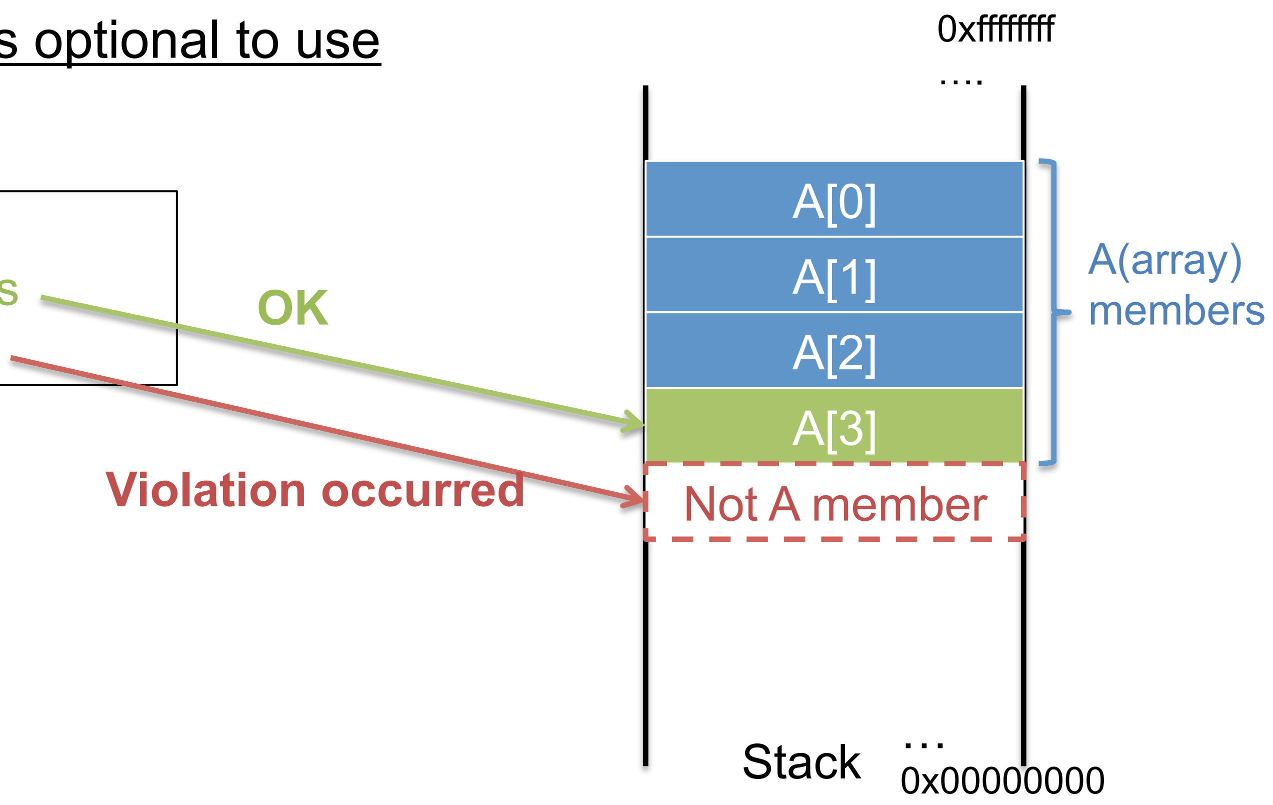
- <u>Intel MPX supported CPU has not yet been shipped at Feb. 25, 2015</u>
  - "skylake" architecture is scheduled to equip with MPX which anticipated in Q3 2015
  - Now, we can test MPX feature on Intel Software Development Emulator only

* **Intel** are trademarks of Intel Corporation in the U.S. and/or other countries in this documents

A Real Application Example: Memory Protection with Code Instrumentation

1. GCC Compiler is ready for memory protection using Intel MPX
2. GCC's memory protection support was merged into Linux kernel 3.19
   – Kernel support is optional to use

```
int A[4];
A[3] = 0; // Regular access
A[4] = 0; // Illegal access
```



Intel Memory Protection Extensions(Intel MPX)

Intel MPX Overview

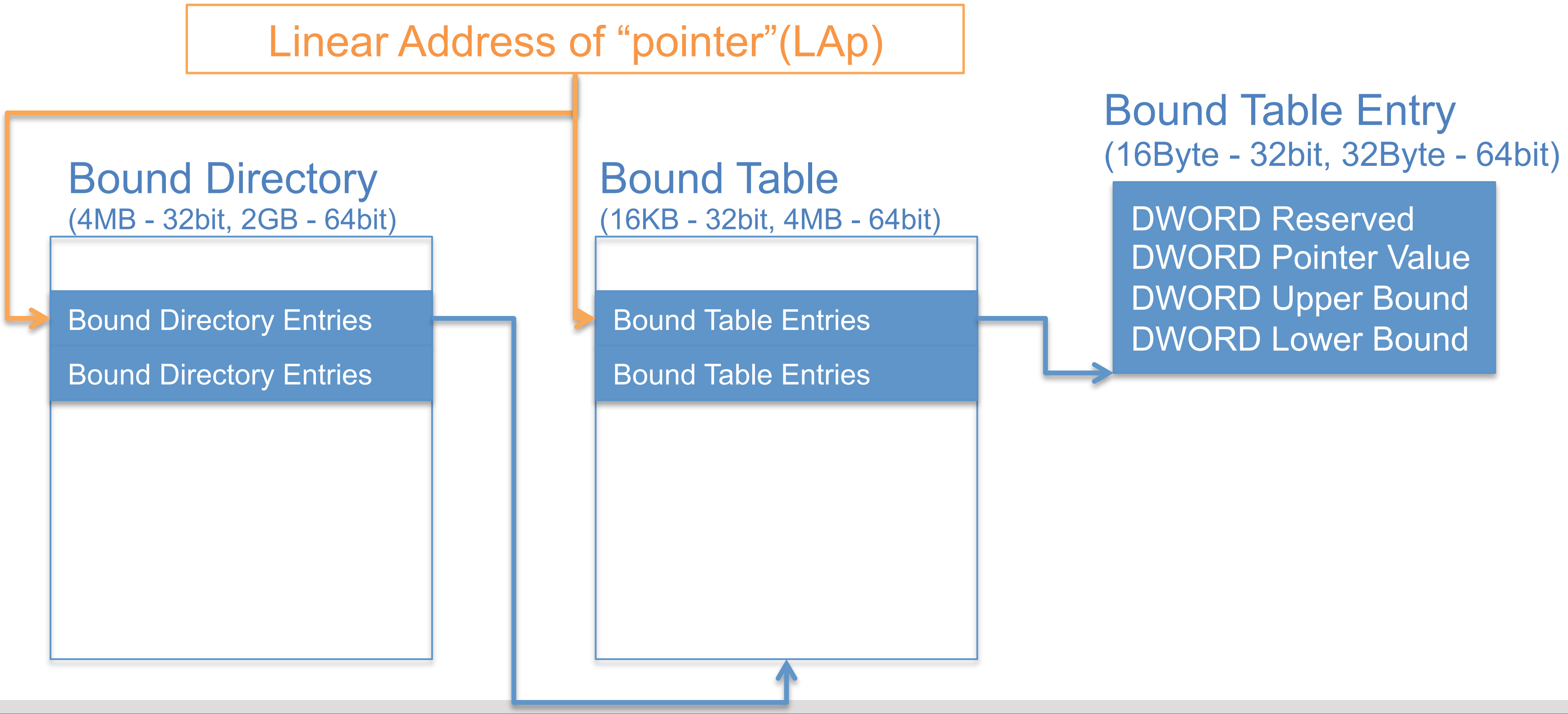
- Added new registers and instruction set
  - To achieve an interoperability, new instruction’s prefix is "0x”.
- Added bound paging
  - Bound paging associates address pointer with memory bounds
- Bounds registers are saving and restoring same as the another CPU context switching
  - e.g. CALL, RET, JMP and Jcc
- VMX and TSX instruction set were extended

New Registers

- BND0-3 Bounds registers
  - 128bit registers
  - Bounds register saves pair of 64-bit values
- BNDCFGU register
  - Configuration register for bound paging in Ring 3
- BNDCFGS register
  - Configuration register for bound paging in Ring 0, 1, 2
- BNDSTATUS register
  - Saving bounds check status

Bound Paging

- Bounds mapping management mechanism like memory paging
- Bounds paging requires to allocate linear address space



BND Instruction Set

- BNDMK
  - Saving bound pair for bounds register
- BNDCL
  - Checking lower bound with bounds register lower value and operands
- BNDCU, BNDCN
  - Checking upper bound with bounds register upper value and operands
- BNDMOV
  - Accessing bounds register
- BNDLDX
  - Loading bounds register from memory
- BNDSTX
  - Saving bounds register to memory

Use case (1)

```
//Assuming  that integer array preserves under stack base
; int A[100] from RBP

// Storing base address of array in RAX register
 LEA RAX, [RBP+offset]

// Storing the second operand in lower bound(0-63bit)
// and upper bound(127-64bit) of BND0 register
BNDMK BND0, [RAX+399]
```

Use case (2)

```
//Storing RBX value in RAX address
BNDCL BND0, [RAX] ;Lower bound check
BNDCU BND0, [RAX+3] ; Upper bound check
MOV Dword ptr [RAX], RBX;
```

Exception #BR has occurred and sets error code to BNDSTATUS register if bound check detects memory access violation

Error Code in BNDSTATUS

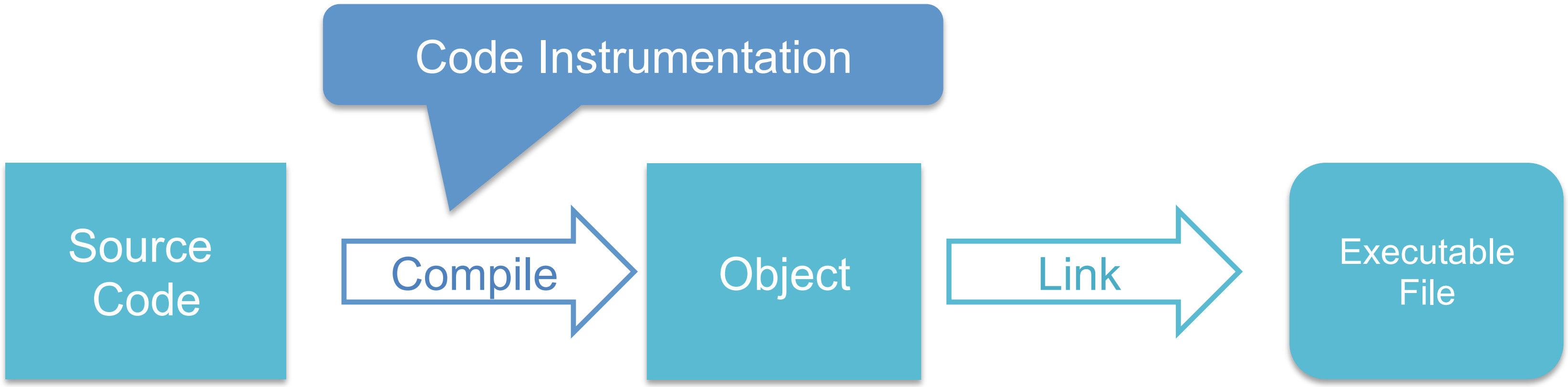
- BNDSTATUS register saves error code
  - 00b – NO Intel MPX exception
  - 01b – Bounds violation
    - BNDCL, BNDLU, BNDCN raises this error code
  - 10b – Invalid bound directory entry
    - BNDLDX BNDSTX raises this error code
    - Invalid bound directory entry address is saved in BNDSTATUS(127-2bit)
  - 11b - reserved

Intel MPX Support in GCC Compiler

Intel MPX support in the GCC compiler

- GCC compiler provides memory protection using Intel MPX now
  - "-f mpx" option

- Memory protection is achieved via code instrumentation



Code Instrumentations by “–f mpx”

- Bounds computation
  - Bounds are computed in accordance with compile time
- Memory access instrumentation
  - Compiler detects all memory accesses and inserts code to checks with bounds
- Calls&Returns instrumentation
- Bounds table management
- Narrowing

Reference：
https://gcc.gnu.org/wiki/Intel%20MPX%20support%20in%20the%20GCC%20compiler

Example: Memory Accesses Instrumentation

- It shows instrumented code (put zero into p[i])

```
p[i] = 0;
```

$\xrightarrow{\text{Instrumentation}}$

```
<bb 2>:
  _2 = (long unsigned int) i_1(D);
  _3 = _2 * 4;
  _6 = p_4(D) + _3;
  __builtin_ia32_bndcl (__bounds_of_p_5, _6);
  _8 = _6 + 3;
  __builtin_ia32_bndcu (__bounds_of_p_5, _8);
  *_6 = 0;
```

Reference：
https://gcc.gnu.org/wiki/Intel%20MPX%20support%20in%20the%20GCC%20compiler

Intel MPX in Linux Kernel

Intel MPX in Linux Kernel

- Intel MPX support was merged into Linux kernel 3.19
  - Bounds table tracing for dynamic allocation by mmap and munmap
  - Added bound pair into extended struct siginfo
  - Added two prctl commands for enable and disable bounds table management

Bounds Table Tracing for Dynamic Allocation

- Bounds table requires to allocate on linear address space
  - In 64bit mode, bound table max size is 2GB

- Kernel supports dynamic allocation of bounds table for memory allocation effectiveness
  - Added VM_MPX flag to VM page attributes

Extended siginfo structure

- Added bounds violation fields to _sigfault

```
/* SIGILL, SIGFPE, SIGSEGV, SIGBUS */
struct {
        void __user *_addr; /* faulting insn/memory ref. */
        #ifdef __ARCH_SI_TRAPNO
        int _trapno;        /* TRAP # which caused the signal */
        #endif
        short _addr_lsb; /* LSB of the reported address */
        struct {
                void __user *_lower;
                void __user *_upper;
        } _addr_bnd;
} _sigfault;
```

in include/uapi/asm-generic/siginfo.h in Linux kernel source (3.19)

Conclusion

- Intel MPX provides primitive functions for runtime memory protection via compiler’s code instrumentation

- Performance impact is not clear
  - However, we guess it is faster than another approach of memory protection such as software fault isolation or runtime instrumentation

- After all, Intel MPX and runtime memory protection are not expected to come into wide use for some time because we need replacement our desktops and servers for to use buffer overflow protection

Bibliography

- Intel® Architecture Instruction Set Extensions Programming Reference Section 9 Version 319433-022 OCTOBER 2014
- Intel® Memory Protection Extensions (Intel® MPX) support in the GCC compiler https://gcc.gnu.org/wiki/Intel%20MPX%20support%20in%20the%20GCC%20compiler
- Documentations/x86/intel-mpx.txt in Linux kernel 3.19

Contact Information

E-Mail : research—feedback@ffri.jp

Twitter : @FFRI_Research