(12) 특허협력조약에 의하여 공개된 국제출원

(19) 세계지식재산권기구
국제사무국

(43) 국제공개일
2024 년 3 월 7 일 (07.03.2024)

WIPO | PCT

(10) 국제공개번호
WO 2024/048922 A1

(51) 국제특허분류:
G06F 21/55 (2013.01) *G06F 21/53* (2013.01)
G06F 21/57 (2013.01)

(21) 국제출원번호: PCT/KR2023/007599

(22) 국제출원일: 2023 년 6 월 2 일 (02.06.2023)

(25) 출원언어: 한국어

(26) 공개언어: 한국어

(30) 우선권정보:
10-2022-0110820 2022 년 9 월 1 일 (01.09.2022) KR

(71) 출원인: 울산과학기술원 (UNIST(ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY)) [KR/KR]; 44919 울산광역시 울주군 언양읍 유니스트길 50, Ulsan (KR).

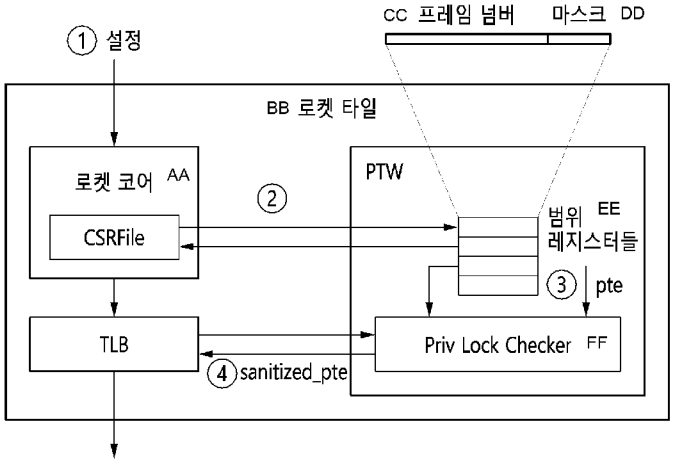
(72) 발명자: 문현곤 (MOON, Hyungon); 44919 울산광역시 울주군 언양읍 유니스트길 50, Ulsan (KR). 하선 (HA, Seon); 44919 울산광역시 울주군 언양읍 유니스트길 50, Ulsan (KR).

(74) 대리인: 특허법인 무한 (MUHANN PATENT & LAW FIRM); 06144 서울특별시 강남구 언주로 560, 8층, Seoul (KR).

(81) 지정국 (별도의 표시가 없는 한, 가능한 모든 종류의 국내 권리의 보호를 위하여): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CV, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IQ, IR, IS, IT, JM, JO, JP, KE, KG, KH, KN, KP, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, MG, MK, MN, MU, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW,

(54) **Title:** ELECTRONIC DEVICE AND METHOD FOR CONTROLLING OPERATION OF KERNEL CODE AREA

(54) **발명의 명칭:** 커널 코드 영역에 대한 동작을 제어하기 위한 전자 장치 및 방법



① ... Configure
AA ... Rocket core
BB ... Rocket tile
CC ... Frame number
DD ... Mask
EE ... Range registers
FF ... Priv Lock Checker

(57) **Abstract:** An electronic device for controlling the operation of a kernel code area according to an embodiment may include a processor which: in machine mode, configures a range register on the basis of the physical address of the kernel code area in which a kernel code is stored; in kernel mode, uses a page table entry for a virtual address of a page to acquire the physical address of the page in response to a case in which the processor receives an access request based on the virtual address of the page; determines whether an area corresponding to the acquired physical address is included in the kernel code area by comparing the configured range register with the acquired physical address; and determines whether to enable or disable at least one of a write operation or an execution operation for the area corresponding to the physical address on the basis of whether the area corresponding to the acquired physical address is

WO 2024/048922 A1

[다음 쪽 계속]

SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

(84) **지정국** (별도의 표시가 없는 한, 가능한 모든 종류의 역내 권리의 보호를 위하여): ARIPO (BW, CV, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SC, SD, SL, ST, SZ, TZ, UG, ZM, ZW), 유라시아 (AM, AZ, BY, KG, KZ, RU, TJ, TM), 유럽 (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, ME, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

공개:

— 국제조사보고서와 함께 (조약 제21조(3))

included in the kernel code area.

(57) 요약서: 일 실시예에 따른 커널 코드 영역에 대한 동작을 제어하기 위한 전자 장치는, 프로세서가 머신 모드(machine mode)인 동안, 커널 코드(kernel code)가 저장된 커널 코드 영역의 물리 주소(physical address)에 기초하여 범위 레지스터(range register)를 설정(configure)하고, 상기 프로세서가 커널 모드(kernel mode)인 동안 페이지(page)의 가상 주소에 기초한 액세스 요청을 수신하는 경우에 응답하여, 상기 페이지의 상기 가상 주소에 대한 페이지 테이블 엔트리(page table entry)를 이용하여 상기 페이지의 물리 주소를 획득하며, 상기 설정된 범위 레지스터를 상기 획득된 물리 주소와 비교함으로써, 상기 획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는지 여부를 판단하고, 상기 획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는지 여부에 기초하여, 상기 물리 주소에 대응하는 영역에 대한 쓰기 동작 또는 실행 동작 중 적어도 하나의 동작에 대한 인에이블(enable) 또는 디스에이블(disable) 여부를 결정하는 프로세서를 포함할 수 있다.

명세서

발명의 명칭: 커널 코드 영역에 대한 동작을 제어하기 위한 전자 장치 및 방법

기술분야

[1] 이하, 커널 코드가 저장된 메모리 영역에 대한 동작을 제어하는 기술이 개시된다.

배경기술

[2] 운영 체제 커널은 다양한 레벨들에서 시스템을 관리하는 권한 있는 소프트웨어 구성 요소를 포함할 수 있다. 사용자 프로세스들과 긴밀하게 상호 작용하고 대부분의 리소스들을 관리하는 운영 체제 커널에 대한 공격은, 공격자에게 전체 시스템을 강력한 제어를 수여할 수 있다. 커널이 실행하는 코드는 커널의 중심에 존재할 수 있다. 코드는 커널이 동작하는 방법을 정의하므로, 런타임에 온전하게 유지되어야 할 수 있다. 커널 코드 무결성은 커널이 실행하는 코드가 런타임에 온전하게 유지되어야 하는 특성을 나타낼 수 있다. 커널 코드 무결성은 공격자에 의하여 두 가지 방법으로 위반될 수 있다.

[3] 예를 들어, 커널 코드 무결성은 공격자에 의한 커널 코드의 수정을 통해 위반될 수 있다. 프로세서는 커널의 권한으로 수정된 코드를 실행하도록 만들 수 있다. 예를 들어, 일부 시스템 호출 핸들러들이 덮어씌워짐으로써, 공격자는 커널이 대상 시스템 호출을 핸들링하는 방식을 변경할 수 있다.

[4] 예를 들어, 커널 코드 무결성은 커널 코드가 포함되지 않은 새로운 위치(예: 데이터 페이지)로부터의 실행을 통해 위반될 수 있다. 커널은 일반적으로 이 새로운 위치로 점프하지 않을 수 있지만, 공격자는 커널이 간접적인 점프 대상들을 결정하기 위하여 이용하는 리턴 주소들 또는 함수 포인터들과 같은, 코드에 대한 하나 이상의 포인터들을 조작할 수 있다.

[5] 공격자는 커널 코드 무결성을 위반할 인센티브가 존재할 수 있다. 인센티브는 피해자 시스템을 조작할 수 있는 높은 레벨의 자유를 제공하는 것을 포함할 수 있다. 예를 들어, iOS 기기에 대한 제일브레이크 툴(jailbreak tool), 일반적으로 기기가 전술한 페이지 테이블 기반 보호로 보호되지 않을 때, Apple 앱 스토어의 외부로부터 어플리케이션을 설치하도록 커널을 패치할 수 있다. 일부 사용자 레벨 프로그램들이 기기가 Android 플랫폼에 기반하는지 여부를 결정하는 것을 시도하는 경우에도, 이러한 공격자는 커널에 코드를 삽입함으로써 확인을 쉽게 무효화(nullify)할 수 있다.

[6] 컴퓨터 시스템의 보안의 초석으로 간주되고 있음에도 불구하고, 기존 및 배포된 많은 보호 메커니즘들은 공격에 취약할 수 있다. 예를 들어, Microsoft Windows는 KPP(Kernel Patch Protection)라는 기능을 가질 수 있고, 이전 iOS는 또한 유사한 기능을 가질 수 있다. 기존 및 배포된 많은 보호 메커니즘들은, 커널 코

드 페이지의 스냅샷을 주기적으로 검사함으로써 알 수 없는 취약점을 악용한 공격자에 의하여 커널 코드 내용이 수정되지 않는 것을 보장할 수 있다. 불행히도, 이러한 방어를 알고 있는 공격자는 확인들 사이에서만 커널을 수정함으로써 방어를 우회할 수 있다.

[7] 이러한 조작을 물리칠 수 있는 보다 더 강력한 메커니즘은 페이지 테이블 속성들을 사용하는 것이지만, 보다 더 권한이 있는 소프트웨어 레이어(예: 하이퍼바이저)에 의존함으로써 또는 페이지 테이블 관리의 유연성을 희생하여, 페이지 테이블의 무결성이 보장되는 것을 요구할 수 있다. 예를 들어, Hypervision 및 SPROBES는 TrustZone에 의해 보호되는 Secure Monitor 및 Secure OS에 의존할 수 있고, HVCI는 하이퍼바이저에 의존할 수 있다. KTRR에 대한 세부사항들은 공개되지 않았지만, 메커니즘은 커널 페이지 테이블에 대한 업데이트를 제한함으로써 커널의 가상 주소 공간을 관리하는 유연성을 제한할 수 있다.

[8] RISC-V ISA 사양은 물리적 주소 기반 액세스 제어의 형태를 구현하기 위한 표준들을 이미 포함하고, 작업 그룹은 향상된 버전을 준비하는 중일 수 있다. 이 확장은 물리적 메모리 보호(physical memory protection; PMP)라고 불리고, 주로 수퍼바이저 모드(예: 커널 모드) 프로그램 및 사용자 모드 프로그램이 머신 모드 프로그램의 주소에 액세스하는 것을 금지하도록 설계될 수 있다. 이 표준은 최대 16개의 다른 물리적 주소 범위들 및 대응하는 액세스 제어 정책을 정의할 수 있는 4개의 구성 레지스터들을 정의할 수 있다. 16개의 범위들의 우선 순위는, 16개의 정책들 중 최대 하나가 하나의 물리적 주소에 적용되도록, 엄격하게 지정될 수 있다.

[9] 물리적 메모리 보호(PMP)는 머신 레이어 소프트웨어를 보호하는 데 유용한 것으로 입증되었지만, 커널 코드 무결성을 보호하는 데 사용될 수 없을 수 있다. 물리적 메모리 보호(PMP)는 읽기 가능한(readable; R), 쓰기 가능한(writable; W) 및 실행 가능한(executable; X) 허가들(permissions)을 나타내는 3비트를 각 주소 범위에 연관시킬 수 있고, 이는 실행 전용 물리적 메모리 영역들을 생성하는 것을 가능하게 할 수 있다. 운영 체제 커널은, 머신 모드 소프트웨어가 이러한 속성들을 사용하여 코드 페이지들을 보호하기 위해 코드 페이지를 실행 전용으로 만드는 것을 요청할 수 있습니다. 그러나, 커널은 커널 코드 페이지들의 외부로부터 실행하도록 커널 스스로를 속이는 것을 방지할 수 없다. 물리적 메모리 보호(PMP)는 프로세서가 현재 실행 중인 모드를 고려하지 않고 특정 물리적 주소에 대하여 정의한 16개의 정책들 중 하나만 적용하기 때문일 수 있다. 결과적으로, 수퍼바이저 모드에서 실행되는 운영 체제 커널이 사용자 프로그램의 코드 페이지로부터 코드를 실행하는 것을 금지하는 것은 가능하지 않을 수 있다. 물리적 메모리 보호(PMP)는 수퍼바이저 모드 및 사용자 모드 코드 모두가 물리적 주소 범위로부터 실행되는 것을 허용해야 하기 때문일 수 있다. 물리적 메모리 보호(PMP)의 주요 목표는 나머지로부터 머신 모드 레이어를 보호하는 것이기 때문에, 이는 물리적 메모리 보호(PMP)의 결함이 아닐 수 있다.

[10] 현재 버전의 물리적 메모리 보호(PMP)는 머신 모드에서도 코드 무결성을 보호하는 것이 가능하지 않을 수 있다. 현재 물리적 메모리 보호(PMP)의 사양에 따르면, 하나의 허가만이 주소 일치 사양으로 인하여, 각 주소와 연관될 수 있다. 이러한 이유로 수퍼바이저 및 사용자 프로그램들의 주소 범위들은, 해당 페이지들에 대한 액세스 제어를 수퍼바이저 레벨 프로그램에 위임(delegate)하기 위하여 실행 가능한 것으로 마킹되어야 할 수 있다. 이를 통해, 해당 페이지는 머신 모드에서도 실행될 수 있다. 결과적으로, 머신 모드의 취약점은, 프로세서가 머신 모드에서 실행되는 동안 공격자가 사용자의 메모리에 위치한 페이로드를 실행할 수 있다는 것을 포함할 수 있다. 이러한 한계를 극복하기 위해, 향상된 사양의 초안은, 머신 모드에서 외부 코드의 실행을 위협으로 간주하고 일 실시예에 따른 프로세서와 유사한 사양을 포함할 수 있다. 새로운 사양은, 머신 모드에서 구성된 주소 범위에 속하지 않는 주소를 실행할 수 없도록 만들 수 있다. 따라서, 프로세서가 주소 범위 레지스터들을 사용하여 실행 가능한 코드 페이지의 허용 목록(allowlist)을 정의하는 반면 레지스터는 현재 물리적 메모리 보호(PMP)의 거부 목록들(denylists)만 될 수 있다.

발명의 상세한 설명

과제 해결 수단

[11] 일 실시예에 따른 프로세서에 의하여 수행되는 커널 코드 영역에 대한 동작을 제어하기 위한 방법은, 프로세서가 머신 모드(machine mode)인 동안, 커널 코드(kernel code)가 저장된 커널 코드 영역의 물리 주소(physical address)에 기초하여 범위 레지스터(range register)를 설정(configure)하는 단계를 포함할 수 있다. 일 실시예에 따른 프로세서에 의하여 수행되는 커널 코드 영역에 대한 동작을 제어하기 위한 방법은, 상기 프로세서가 커널 모드(kernel mode)인 동안 페이지(page)의 가상 주소에 기초한 액세스 요청을 수신하는 경우에 응답하여, 상기 페이지의 상기 가상 주소에 대한 페이지 테이블 엔트리(page table entry)를 이용하여 상기 페이지의 물리 주소를 획득하는 단계를 포함할 수 있다. 일 실시예에 따른 프로세서에 의하여 수행되는 커널 코드 영역에 대한 동작을 제어하기 위한 방법은, 상기 설정된 범위 레지스터를 상기 획득된 물리 주소와 비교함으로써, 상기 획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는지 여부를 판단하는 단계를 포함할 수 있다. 일 실시예에 따른 프로세서에 의하여 수행되는 커널 코드 영역에 대한 동작을 제어하기 위한 방법은, 상기 획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는지 여부에 기초하여, 상기 물리 주소에 대응하는 영역에 대한 쓰기 동작 또는 실행 동작 중 적어도 하나의 동작에 대한 인에이블(enable) 또는 디스에이블(disable) 여부를 결정하는 단계를 포함할 수 있다.

[12] 상기 쓰기 동작 또는 실행 동작 중 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정하는 단계는, 상기 획득된 물리 주소에 대응하는 영역이 커

널 코드 영역에 포함되는 경우에 응답하여, 상기 물리 주소에 대응하는 영역에 대한 쓰기 동작(write operation)을 디스에이블하는 단계를 포함할 수 있다.

[13] 상기 쓰기 동작 또는 실행 동작 중 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정하는 단계는, 상기 획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는 경우에 응답하여, 상기 프로세서가 커널 모드인 동안 상기 물리 주소에 대응하는 영역에 대한 실행 동작(execute operation)을 인에이블하는 단계를 포함할 수 있다.

[14] 상기 쓰기 동작 또는 실행 동작 중 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정하는 단계는, 상기 획득된 물리 주소에 대응하는 영역이 커널 코드 영역과 다른 영역에 포함되는 경우에 응답하여, 상기 프로세서가 커널 모드인 동안 상기 물리 주소에 대응하는 영역에 대한 실행 동작(excute operation)을 디스에이블(disable)하는 단계를 포함할 수 있다.

[15] 상기 쓰기 동작 또는 실행 동작 중 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정하는 단계는, 상기 페이지 테이블 엔트리의 페이지 테이블 속성(page table attribute)과 독립적으로, 상기 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정하는 단계를 포함할 수 있다.

[16] 상기 범위 레지스터를 설정하는 단계는, 상기 프로세서가 머신 모드인 동안, 커널 코드를 가지는 페이지를 미리 결정된 크기를 가지는 청크(chunk)의 경계(boundary)에 따라 정렬(align)하는 단계를 포함할 수 있다.

[17] 상기 범위 레지스터를 설정하는 단계는, 상기 커널 코드를 적어도 하나의 커널 코드 청크(kernel code chunk)로 저장하는 단계를 포함할 수 있다. 상기 범위 레지스터를 설정하는 단계는, 각 커널 코드 청크(kernel code chunk)에 대하여, 해당 커널 코드 청크에 대응하는 범위 레지스터의 적어도 일부를 설정하는 단계를 포함할 수 있다.

[18] 상기 범위 레지스터를 설정하는 단계는, 각 커널 코드 청크의 시작 물리 주소에 기초하여, 해당 커널 코드 청크에 대응하는 범위 레지스터의 베이스 비트(base bit)를 설정하는 단계를 포함할 수 있다. 상기 범위 레지스터를 설정하는 단계는, 해당 커널 코드 청크의 크기에 기초하여, 해당 커널 코드 청크에 대응하는 범위 레지스터의 마스크 비트(mask bit)를 설정하는 단계를 포함할 수 있다.

[19] 상기 범위 레지스터를 설정하는 단계는, 상기 범위 레지스터의 베이스 비트 및 마스크 비트를 설정한 경우에 응답하여, 상기 범위 레지스터의 밸리드 비트(valid bit) 및 락 비트(lock bit)를 설정하는 단계를 포함할 수 있다.

[20] 상기 페이지의 물리 주소를 획득하는 단계는, 변환 색인 버퍼(translation lookaside buffer; TLB)에 상기 액세스 요청의 상기 가상 주소에 대한 상기 페이지 테이블 엔트리가 캐시(cache)되어 있는지 여부를 확인하는 단계를 포함할 수 있다. 일 실시예에 따른 프로세서에 의하여 수행되는 커널 코드 영역에 대한 동작을 제어하기 위한 방법은, 상기 변환 색인 버퍼에 상기 페이지 테이블 엔트리가

캐시되어 있는 경우에 응답하여, 상기 설정된 범위 레지스터와 상기 획득된 물리 주소의 비교를 생략하는 단계를 포함할 수 있다.

[21] 일 실시예에 따른 프로세서에 의하여 수행되는 커널 코드 영역에 대한 동작을 제어하기 위한 방법은, 변환 색인 버퍼에 상기 페이지 테이블 엔트리가 캐시되어 있는 경우에 응답하여, 상기 페이지 테이블 엔트리의 페이지 테이블 속성에 기초하여 상기 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정하는 단계를 포함할 수 있다.

[22] 일 실시예에 따른 프로세서에 의하여 수행되는 커널 코드 영역에 대한 동작을 제어하기 위한 방법은, 상기 결정된 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부에 기초하여, 상기 페이지 테이블 엔트리의 페이지 테이블 속성을 업데이트하는 단계를 포함할 수 있다. 일 실시예에 따른 프로세서에 의하여 수행되는 커널 코드 영역에 대한 동작을 제어하기 위한 방법은, 상기 업데이트된 페이지 테이블 속성을 상기 페이지 테이블 엔트리와 함께 변환 색인 버퍼에 캐시하는 단계를 포함할 수 있다.

[23] 상기 획득된 물리 주소에 대응하는 영역 커널 코드 영역에 포함되는지 여부를 판단하는 단계는, 상기 범위 레지스터의 마스크 비트에 기초하여, 상기 물리 주소 중 부분 주소를 결정하는 단계를 포함할 수 있다.

[24] 상기 획득된 물리 주소에 대응하는 영역 커널 코드 영역에 포함되는지 여부를 판단하는 단계는, 상기 결정된 부분 주소가 상기 범위 레지스터의 베이스 비트의 적어도 일부에 매칭되는 경우 상기 획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는 것을 판단하는 단계를 포함할 수 있다.

[25] 일 실시예에 따른 커널 코드 영역에 대한 동작을 제어하기 위한 전자 장치는, 프로세서가 머신 모드(machine mode)인 동안, 커널 코드(kernel code)가 저장된 커널 코드 영역의 물리 주소(physical address)에 기초하여 범위 레지스터(range register)를 설정(configure)하고, 상기 프로세서가 커널 모드(kernel mode)인 동안 페이지(page)의 가상 주소에 기초한 액세스 요청을 수신하는 경우에 응답하여, 상기 페이지의 상기 가상 주소에 대한 페이지 테이블 엔트리(page table entry)를 이용하여 상기 페이지의 물리 주소를 획득하며, 상기 설정된 범위 레지스터를 상기 획득된 물리 주소와 비교함으로써, 상기 획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는지 여부를 판단하고, 상기 획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는지 여부에 기초하여, 상기 물리 주소에 대응하는 영역에 대한 쓰기 동작 또는 실행 동작 중 적어도 하나의 동작에 대한 인에이블(enable) 또는 디스에이블(disable) 여부를 결정하는 프로세서를 포함할 수 있다.

[26] 상기 프로세서는, 상기 쓰기 동작 또는 실행 동작 중 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정할 수 있다. 상기 프로세서는, 상기 획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는 경우에 응답하여, 상기 물리 주소에 대응하는 영역에 대한 쓰기 동작(write operation)을 디스에이블할 수 있다.

[27] 상기 프로세서는, 상기 쓰기 동작 또는 실행 동작 중 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정할 수 있다. 상기 프로세서는, 상기 획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는 경우에 응답하여, 상기 프로세서가 커널 모드인 동안 상기 물리 주소에 대응하는 영역에 대한 실행 동작(execute operation)을 인에이블할 수 있다.

[28] 상기 프로세서는, 상기 쓰기 동작 또는 실행 동작 중 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정할 수 있다. 상기 프로세서는, 상기 획득된 물리 주소에 대응하는 영역이 커널 코드 영역과 다른 영역에 포함되는 경우에 응답하여, 상기 프로세서가 커널 모드인 동안 상기 물리 주소에 대응하는 영역에 대한 실행 동작(excute operation)을 디스에이블(disable)할 수 있다.

[29] 상기 프로세서는, 상기 쓰기 동작 또는 실행 동작 중 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정할 수 있다. 상기 프로세서는, 상기 페이지 테이블 엔트리의 페이지 테이블 속성(page table attribute)과 독립적으로, 상기 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정할 수 있다.

[30] 상기 프로세서는, 상기 프로세서가 머신 모드인 동안, 커널 코드를 가지는 페이지를 미리 결정된 크기를 가지는 청크(chunk)의 경계(boundary)에 따라 정렬(align)할 수 있다.

[31] 상기 프로세서는, 상기 커널 코드를 적어도 하나의 커널 코드 청크(kernel code chunk)로 저장할 수 있다. 상기 프로세서는, 각 커널 코드 청크(kernel code chunk)에 대하여, 해당 커널 코드 청크에 대응하는 범위 레지스터의 적어도 일부를 설정할 수 있다.

[32] 상기 프로세서는, 각 커널 코드 청크의 시작 물리 주소에 기초하여, 해당 커널 코드 청크에 대응하는 범위 레지스터의 베이스 비트(base bit)를 설정할 수 있다. 상기 프로세서는, 해당 커널 코드 청크의 크기에 기초하여, 해당 커널 코드 청크에 대응하는 범위 레지스터의 마스크 비트(mask bit)를 설정할 수 있다.

[33] 상기 프로세서는, 상기 범위 레지스터의 베이스 비트 및 마스크 비트를 설정한 경우에 응답하여, 상기 범위 레지스터의 밸리드 비트(valid bit) 및 락 비트(lock bit)를 설정할 수 있다.

[34] 상기 프로세서는, 변환 색인 버퍼(translation lookaside buffer; TLB)에 상기 액세스 요청의 상기 가상 주소에 대한 상기 페이지 테이블 엔트리가 캐시(cache)되어 있는지 여부를 확인할 수 있다. 상기 프로세서는, 상기 변환 색인 버퍼에 상기 페이지 테이블 엔트리가 캐시되어 있는 경우에 응답하여, 상기 설정된 범위 레지스터와 상기 획득된 물리 주소의 비교를 생략할 수 있다.

[35] 상기 프로세서는, 변환 색인 버퍼에 상기 페이지 테이블 엔트리가 캐시되어 있는 경우에 응답하여, 상기 페이지 테이블 엔트리의 페이지 테이블 속성에 기초하여 상기 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정할 수 있다.

[36] 상기 프로세서는, 상기 변환 색인 버퍼에 상기 페이지 테이블 엔트리가 캐시되어 있지 않은 경우에 응답하여, 페이지 테이블로부터 상기 가상 주소에 대한 상기 페이지 테이블 엔트리를 검색할 수 있다. 상기 프로세서는, 상기 검색된 페이지 테이블 엔트리에 기초하여, 상기 가상 주소에 매핑된 상기 페이지의 상기 물리 주소를 획득할 수 있다.

[37] 상기 프로세서는, 상기 결정된 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부에 기초하여, 상기 페이지 테이블 엔트리의 페이지 테이블 속성을 업데이트할 수 있다. 상기 프로세서는, 상기 업데이트된 페이지 테이블 속성을 상기 페이지 테이블 엔트리와 함께 변환 색인 버퍼에 캐시할 수 있다.

[38] 상기 프로세서는, 상기 범위 레지스터의 마스크 비트에 기초하여, 상기 물리 주소 중 부분 주소를 결정할 수 있다. 상기 프로세서는, 상기 결정된 부분 주소가 상기 범위 레지스터의 베이스 비트의 적어도 일부에 매칭되는 경우 상기 획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는 것을 판단할 수 있다.

발명의 효과

[39] 커널 코드 무결성은 커널 페이지 테이블 무결성을 먼저 보호하지 않고도 보장될 수 있다. 프로세서의 액세스 제어 논리를 재사용함으로써 모든 메모리 액세스들을 검사하는 작은 하드웨어 확장은 도입될 수 있다.

[40] 일 실시예에 따른 프로세서는 성능과 에너지 소비뿐만 아니라 칩 면적과 임계 경로 지연에도 작은 영향을 미치도록 설계되고 구현될 수 있다. 메커니즘은 포괄적으로 평가될 수 있다.

[41] 일 실시예에 따른 프로세서는 제안된 마이크로아키텍처 변경 사항들을 평가하기 위하여, RTL에서 RISC-V SoC, Rocket Chip의 구현을 확장함으로써 구현될 수 있고, FPGA 기반 프로토타입에 대한 실험으로 평가될 수 있다.

도면의 간단한 설명

[42] 도 1은 일 실시예에 따른 프로세서의 동작을 설명하기 위한 도면이다.

[43] 도 2는 일 실시예에 따른 프로세서가 커널 코드 영역에 대한 동작을 제어하기 위한 방법을 설명하기 위한 도면이다.

[44] 도 3은 일 실시예에 따른 범위 레지스터의 설정 동작을 설명하기 위한 도면이다.

[45] 도 4는 일 실시예에 따른 복수의 비트들을 가지는 범위 레지스터의 구조를 설명하기 위한 도면이다.

[46] 도 5는 일 실시예에 따른 프로세서가 액세스 요청의 가상 주소에 매핑된 물리 주소를 획득하는 동작을 설명하기 위한 도면이다.

[47] 도 6는 일 실시예에 따른 프로세서가 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는지 여부를 판단하는 동작을 설명하기 위한 도면이다.

[48] 도 7는 일 실시예에 따른 프로세서가 물리 주소에 대응하는 영역에 대한 동작의 인에이블 또는 디스에이블 여부를 결정하는 동작을 설명하기 위한 도면이다.

[49] 도 8은 일 실시예에 따른 프로세서가 액세스 요청의 가상 주소에 대한 페이지 테이블 엔트리가 변환 색인 버퍼에 캐시되어 있지 않은 경우 페이지 테이블 속성을 업데이트하는 동작을 설명하기 위한 도면이다.

[50] 도 9은 일 실시예에 따른 프로세서가 액세스 요청의 가상 주소에 대한 페이지 테이블 엔트리가 변환 색인 버퍼에 캐시되어 있는 경우 페이지 테이블 속성을 업데이트하는 동작을 설명하기 위한 도면이다.

[51] 도 10은 일 실시예에 따른 프로세서 및 비교 실시예에 따른 프로세서의 성능을 LMBench를 이용하여 측정한 결과를 설명하기 위한 도면이다.

[52] 도 11은 일 실시예에 따른 비교 실시예에 따른 프로세서를 가지는 시스템의 실행 시간으로 정규화된 일 실시예에 따른 프로세서를 가지는 시스템의 워크로드의 실행 시간을 도시한다.

[53] 도 12는 일 실시예에 따른 SPEC 2006 벤치마크 제품군의 12개의 워크로드들의 정규화된 실행 시간을 도시한다.

[54] 도 13은 일 실시예에 따른 프로세서의 정규화된 추가적인 에너지 소모를 설명하기 위한 도면이다.

발명의 실시를 위한 최선의 형태

[55] 실시예들에 대한 특정한 구조적 또는 기능적 설명들은 단지 예시를 위한 목적으로 개시된 것으로서, 다양한 형태로 변경되어 구현될 수 있다. 따라서, 실제 구현되는 형태는 개시된 특정 실시예로만 한정되는 것이 아니며, 본 명세서의 범위는 실시예들로 설명한 기술적 사상에 포함되는 변경, 균등물, 또는 대체물을 포함한다.

[56] 제1 또는 제2 등의 용어를 다양한 구성요소들을 설명하는데 사용될 수 있지만, 이런 용어들은 하나의 구성요소를 다른 구성요소로부터 구별하는 목적으로만 해석되어야 한다. 예를 들어, 제1 구성요소는 제2 구성요소로 명명될 수 있고, 유사하게 제2 구성요소는 제1 구성요소로도 명명될 수 있다.

[57] 어떤 구성요소가 다른 구성요소에 "연결되어" 있다고 언급된 때에는, 그 다른 구성요소에 직접적으로 연결되어 있거나 또는 접속되어 있을 수도 있지만, 중간에 다른 구성요소가 존재할 수도 있다고 이해되어야 할 것이다.

[58] 단수의 표현은 문맥상 명백하게 다르게 뜻하지 않는 한, 복수의 표현을 포함한다. 본 명세서에서, "포함하다" 또는 "가지다" 등의 용어는 설명된 특징, 숫자, 단계, 동작, 구성요소, 부분품 또는 이들을 조합한 것이 존재함으로 지정하려는 것이지, 하나 또는 그 이상의 다른 특징들이나 숫자, 단계, 동작, 구성요소, 부분품 또는 이들을 조합한 것들의 존재 또는 부가 가능성을 미리 배제하지 않는 것으로 이해되어야 한다.

[59] 다르게 정의되지 않는 한, 기술적이거나 과학적인 용어를 포함해서 여기서 사용되는 모든 용어들은 해당 기술 분야에서 통상의 지식을 가진 자에 의해 일반적으로 이해되는 것과 동일한 의미를 가진다. 일반적으로 사용되는 사전에 정의되

어 있는 것과 같은 용어들은 관련 기술의 문맥상 가지는 의미와 일치하는 의미를 갖는 것으로 해석되어야 하며, 본 명세서에서 명백하게 정의하지 않는 한, 이상적이거나 과도하게 형식적인 의미로 해석되지 않는다.

[60] 이하, 실시예들을 첨부된 도면들을 참조하여 상세하게 설명한다. 첨부 도면을 참조하여 설명함에 있어, 도면 부호에 관계없이 동일한 구성 요소는 동일한 참조 부호를 부여하고, 이에 대한 중복되는 설명은 생략하기로 한다.

[61] 도 1은 일 실시예에 따른 프로세서의 동작을 설명하기 위한 도면이다.

[62] 일 실시 예에 따르면, 프로세서는 프로세서에서 실행되는 프로그램들에 의한 데이터 액세스를 처리하기 위해 가상 메모리 기술을 사용할 수 있다. 예를 들어, 프로그램들은 응용 프로그램들, 운영 체제들, 장치 드라이버들, 가상 기계들, 주변 장치들에 의해 실행되는 소프트웨어 등을 포함할 수 있다.

[63] 일반적으로, 프로세서 내의 프로그램에 의해 데이터가 액세스될 때, 데이터를 포함하는 소정의 크기(예: 4 kB, 2 MB)의 블록 또는 "페이지"가 대용량 저장 장치(예: 디스크 드라이브, 반도체 메모리)로부터 프로세서 내 메모리에서의 이용 가능한 물리적 위치로 복사되거나 메모리에 새롭게 생성될 수 있다. 프로그램들이 메모리에서의 페이지들의 물리적 위치들을 추적하도록 요구받는 것을 피하기 위해, 프로세서가 프로그램들에 대한 페이지들의 물리적 위치들을 추적할 수 있다. 프로그램들은 페이지들의 물리적 위치들에 기초한 주소들(또는 물리 주소들)을 사용하여 메모리에 액세스하는 대신, 대응하는 프로그램들에 특정한 로컬 주소 공간들인 가상 주소 공간들의 가상 주소들을 사용하여 메모리에 액세스할 수 있다. 프로그램의 관점에서, 가상 주소들은 데이터가 메모리에 저장되는 실제 물리적 위치들을 나타내고, 그에 따라 가상 주소들을 사용하여 프로그램들이 메모리 액세스를 수행할 수 있다. 그러나, 가상 주소들은 메모리의 페이지들에 데이터가 저장되는 실제 위치들의 물리 주소들에 직접 매핑되지 않을 수 있다. 프로세서는 페이지들의 물리적 위치들을 추적하는 일환으로, 메모리 액세스 요청들에서의 프로그램들에 의해 사용되는 가상 주소들을 데이터가 실제로 위치한 물리 주소들로 변환할 수 있다. 그 다음 프로세서는 물리 주소들을 사용하여 프로그램들에 대한 메모리 액세스를 수행할 수 있다.

[64] 일 실시예에 따르면, 프로세서는, 메인 메모리의 물리 주소를 프로그램에 의해 사용되는 가상 메모리 주소로 추상화할 수 있다. 가상 주소와 물리 주소 사이의 변환은 전형적으로, 메모리 관리 유닛(memory management unit; MMU)으로 지칭되는 하드웨어에 의해 수행될 수 있다.

[65] 가상 주소 및 물리 주소 간의 변환을 위해, 메모리 관리 유닛(MMU)은 페이지 테이블을 포함할 수 있다. 페이지 테이블은 페이지 테이블 엔트리(page table entry; PTE)(또는 엔트리(entry))를 메모리에 저장된 데이터의 페이지들에 대한 가상 주소 대 물리 주소 변환 정보와 포함하는 프로세서의 메모리에 저장된 레코드를 포함할 수 있다. 예를 들어, 페이지 테이블은 가상 주소들에 대응하는 물리 주소들로의 매핑들을 가질 수 있다. 페이지 테이블에서의 검색은 페이지 테이

블 워크(page table walk)로 지칭될 수 있고, 페이지 테이블 워커(page table walker; PTW)로서 지칭되는 메모리 관리 유닛(MMU)의 하드웨어에 의해 수행될 수 있다. 예를 들어, 메모리 관리 유닛(MMU)은 소정의 가상 주소의 메모리에 액세스하기 위한 액세스 요청을 프로그램으로부터 수신하면, 페이지 테이블 워크를 수행함으로써 페이지 테이블로부터 대응하는 물리 주소 정보를 획득할 수 있다.

[66] 전술한 페이지 테이블 워크들이 상대적으로 느리기 때문에, 페이지 테이블 워크들을 수행하는 것을 피하는 것이 바람직할 수 있다. 따라서, 프로세서는 페이지 테이블 워크들 동안 획득된 페이지 테이블 엔트리들(또는 페이지 테이블 엔트리들에 기초한 정보)의 제한된 수의 카피를 저장하기 위해 사용되는 로컬 캐시들인 변환 색인 버퍼(translation lookaside buffer; TLB)들을 포함한다. 동작 동안, 프로세서는, 가상 주소 및 물리 주소 간의 변환을 수행하기 위하여, TLB로부터 캐싱된 페이지 테이블 엔트리들을 획득하는 것을 시도할 수 있다.

[67] TLB에는 복수의 페이지 테이블 엔트리들이 저장될 수 있다. 액세스 요청의 가상 주소에 대한 페이지 테이블 엔트리가 TLB 내에 캐시되어 있으면(예: TLB 히트(TLB hit)이면), 해당 변환이 즉시 이용 가능할 수 있다. 액세스 요청의 가상 주소에 대한 페이지 테이블 엔트리가 TLB 내에 캐시되어 있지 않으면(예: TLB 미스(TLB miss)이면), 페이지 테이블 워크를 통해 페이지 테이블 엔트리를 획득하고, 획득된 페이지 테이블 엔트리의 카피를 TLB에 캐싱할 수 있다.

[68] 이하, 본 명세서에서 주로 일 실시예에 따른 커널 코드 영역에 대한 동작을 제어하기 위한 방법을 수행하는 RISC-V 아키텍처 확장에 기초하여 설명한다.

[69] RISC-V는 축소 명령어 집합 컴퓨팅(reduced instruction set computing)(RISC) 명령어 집합 아키텍처(instruction set architecture)(ISA)의 오픈 소스 구현을 포함할 수 있다. 메타데이터 태그는 각 워드에 대한 명령어 및 데이터 둘 모두에 배치된다. RISC-V 아키텍처에서 워드는 64 비트일 수 있다. RISC-V 아키텍처는 상이한 워드 크기 변형성을 제공할 수 있다. 예를 들어, RV64는 64 비트 워드 크기이고 RV32는 32 비트 워드 크기일 수 있다. 레지스터 및 사용자 주소 공간의 너비 또는 크기는 워드 크기에 따라 다를 수 있다. 태그 크기 또는 너비는 워드 크기 또는 너비와 독립적일 수 있지만, 예시적으로, 태그 크기 또는 너비는 워드 크기 또는 너비와 동일할 수 있다.

[70] RISC-V 아키텍처는 예를 들어, ["The RISC-V Instruction Set Manual Vol. I, User-Level ISA, Version 2.0", May 6, 2014, Waterman, Andrew, et. al., ("also referred to as the "RISC-V user level ISA")]에 설명된 바와 같은 사용자 레벨 명령어를 포함하며, 이 문헌은 본 명세서에서 참조로 포함되고, 예를 들어 RISCV.ORG 웹 사이트 및 버클리 소재의 유니버시티 오브 캘리포니아(University of California)를 통해 Technical Report UCB/EECS-2014-54로서 대중에게 입수 가능하다. RISC-V 아키텍처는 또한 예를 들어 ["The RISC-V Instruction Set Manual Volume II: Privileged Architecture, Version 1.7", May 9, 2015, also referred to as the "RISC-V privileged ISA")]에 설명된 바와 같이, 오퍼레이팅 시스템, 부착형 외부 디바이스 등을 실행

하는데 필요한 권한 있는 명령어 및 추가 기능성을 포함하는 권한 있는 아키텍처를 통합하며, 이 문헌은 본 명세서에서 참조로 포함되고, 예를 들어, RISCV.ORG 웹 사이트 및 버클리 소재의 University of California를 통해 Technical Report UCB/EECS-2015-49로서 대중에게 입수 가능하다.

[71] RISC-V 아키텍처의 실시예는 다음과 같은 네 개의 RISC-V 권한 레벨을 가질 수 있다: 사용자/애플리케이션(user/application; U) 권한 레벨에 대한 레벨 0, 수퍼바이저(supervisor; S) 권한 레벨에 대한 레벨 1, 하이퍼바이저(hypervisor; H) 권한 레벨에 대한 레벨 2, 및 머신(machine; M) 권한 레벨에 대한 레벨 3. 프로세서는 사용자 모드(user mode)인 동안, 사용자/애플리케이션 권한 레벨을 가질 수 있다. 프로세서는, 커널 모드(kernel mode)인 동안, 수퍼바이저 권한 레벨을 가질 수 있다. 프로세서는, 머신 모드(machine mode)인 동안, 머신 권한 레벨을 가질 수 있다. RISC-V 권한 레벨은 최고 내지 최저의 0부터 3까지의 순위가 매겨질 수 있고, 레벨 0이 가장 낮은 권한 레벨을 나타내고, 레벨 3이 가장 높은 권한 레벨을 나타낼 수 있다. 이러한 권한 레벨은 상이한 컴포넌트들 간에 보호를 제공하는데 사용될 수 있으며, 현재 권한 레벨 또는 모드에서 허용되지 않는 연산을 수행하는 코드를 실행하려 시도하면 기본 실행 환경으로 향하게 하는 트랩과 같은 예외가 일어나도록 유발할 수 있다.

[72] 머신 레벨(예: 프로세서가 머신 모드인 동안 가지는 권한 레벨)은 가장 높은 권한을 가질 수 있고, RISC-V 하드웨어 플랫폼에서 유일한 강제적 권한 레벨(mandatory privilege level)일 수 있다. 머신 모드(M-모드)에서 실행되는 코드는 머신 구현으로의 액세스 레벨이 낮기 때문에 본질적으로 신뢰성을 가질 수 있다. 사용자 모드(U-모드) 및 수퍼바이저 모드(S-모드)는 각각 통상의 애플리케이션 및 오퍼레이팅 시스템용으로 의도되지만, 하이퍼바이저 모드(H-모드)는 가상 머신 모니터를 지원하도록 의도될 수 있다. 각 권한 레벨은 임의적 확장과 변형성을 가진 핵심 권한 ISA 확장 세트를 가질 수 있다.

[73] RISC-V 아키텍처의 구현은 적어도 M-모드를 지원하여야 하고 대부분의 구현은 적어도 U-모드 및 M-모드를 지원한다는 것을 유의하여야 할 수 있다. S-모드는 수퍼바이저 레벨 오퍼레이팅 시스템의 코드와 M-모드에서 실행되는 다른 보다 권한 있는 코드 간의 추가 분리를 제공하기 위해 추가될 수 있다. 사용자 또는 애플리케이션 코드는 전형적으로 트랩(예를 들어, 수퍼바이저 호출, 페이지 폴트(page fault)) 또는 인터럽트가 발생하여 지원되는 상위 권한 모드 또는 레벨(예를 들어, H, S 또는 M 모드) 중 하나에서 실행되는 트랩 핸들러로 제어를 강제로 이전할 때까지 U-모드에서 실행될 수 있다. 이후 트랩 핸들러의 코드가 실행된 다음 트랩을 유발한 원래 사용자 코드 또는 애플리케이션으로 제어가 반환될 수 있다. 이러한 사용자 코드 또는 애플리케이션의 실행은 트랩 핸들러 호출을 트리거했던 U-모드의 원래의 트랩된 명령어에서 또는 그 이후에서 재개될 수 있다.

[74] RISC-V 구현에서 지원되는 모드들의 다양한 조합은, 예시적으로, 단일 M 모드, M 및 U를 가지는 두 개의 모드들, M, S 및 U를 가지는 세 개의 모드들, 또는 M, H, S, 및 U를 가지는 네 개의 모드들을 포함할 수 있다.

[75] RISC-V 아키텍처는 하나 이상의 연관된 권한 레벨에 의해 판독되고 수정될 수 있는 제어 상태 레지스터(Control Status Register; CSR)를 가질 수 있다. CSR은 네 개의 권한 레벨 중 첫 번째 권한 레벨과 첫 번째 권한 레벨보다 높은 네 개 권한 레벨 중 임의의 다른 권한 레벨에서 액세스 가능할 수 있다. 예를 들어, 프로그램이 U-모드(예: 레벨 0)에서 실행 중일 때 규칙 캐시 미스와 같은 트랩이 발생하면, 제어는 규칙 캐시 미스 핸들러 코드와 같은 상위 권한 또는 모드(예: 레벨 1 내지 3 중 임의의 레벨)에서 실행 중인 트랩 핸들러로 이전될 수 있다. 트랩이 발생하면, 정보는 M-모드에서 실행되는 트랩 핸들러에 액세스 가능한 CSR에 배치될 수 있다. M-모드에서 실행되는 트랩 핸들러에 액세스 가능한 CSR은, 더 낮은 권한 레벨에서 실행되는 임의의 다른 코드(예: H, S 또는 U 모드에서 액세스 불가능한 코드)에 액세스 불가능할 수 있다.

[76] 많은 임베디드 시스템들에서, 운영 체제 커널은 프로세스들 간의 격리(isolation) 및 액세스 제어 정책들의 시행(enforcement)을 포함한 리소스 관리를 책임질 수 있다. 이러한 책임은, 커널이 간결하게 작성되는 것을 어렵게 만들 수 있고, 매년 새로운 보안 취약점들이 발견되는 것을 초래할 수 있다. 보안 취약점들 중 하나를 알고 있는 공격자는 페이지 테이블들 또는 코드를 포함하는 커널의 메모리 페이지들을 임의로 읽거나 수정하기 위한 수단들을 획득할 수 있다.

[77] 강력한 공격자가 우회할 수 있는 정책들 중 하나는 커널 코드 무결성(kernel code integrity)을 포함할 수 있다. 커널 코드 무결성은 운영 체제 커널이 커널에 속하지 않은 코드의 조각들을 실행하도록 속지 않아야 하는 특성을 나타낼 수 있다. 시스템이 이 정책(policy)을 시행하는 데 실패하면, 공격자는 거의 임의적인 방식으로 피해자 시스템을 조작할 수 있는 힘(power)을 얻을 수 있다. 예를 들어, 공격자는 임의의 어플리케이션 레벨의 보안 메커니즘 또는 어플리케이션 배포에 관한 공급자 설치된(vendor-installed) 보안 정책도 우회할 수 있다.

[78] 이러한 위협들로 인해, 이러한 공격들을 방지하기 위한 메커니즘은 연구 커뮤니티 및 장치 제조업체들에 의하여 찾아질 수 있다. 오늘날 대부분의 프로세서들은 운영 체제 커널이 특수 페이지 테이블 속성들(attributes)을 커널 코드를 포함하는 페이지들에 연관시키는 것을 가능하게 할 수 있다. 특수 페이지 테이블 속성들을 이용하여, 프로세서는 커널 권한이 있는 모드(예: 수퍼바이저 모드)에서 실행되는 동안 커널 코드를 수정하거나 커널 코드의 외부에서 실행하려는 모든 시도를 거부할 수 있다. 커널 코드 페이지를 읽기 전용으로 만드는 것은 커널 코드 페이지의 이러한 바람직하지 않은 수정들을 금지하지만, 커널 코드 페이지들의 외부로부터의 실행의 방지는 모든 페이지 테이블들에 정책이 적용되는 것이 요구될 수 있다. 후자의 페이지 테이블 속성은 일반적으로 PXN(Privileged Execute Never) 또는 SMEP(Supervisor Mode Execution Prevention)라고 불릴 수 있다. 프로

세서가 수퍼바이저 모드(예: 커널 권한 레벨을 가지는 커널 모드)에서 실행되는 동안, 설정될 때, 페이지들은 실행 가능하지 않은 것으로 간주될 수 있다. 이는, 원하는 목표를 달성하기 위하여, 커널 코드 페이지용을 제외하고 커널이 생성하는 모든 페이지 테이블 엔트리들은 비트 세트를 가져야 하는 것을 의미할 수 있다. 그럼에도 불구하고, 최신 운영 체제 커널들은, 커널에 의하여 승인된 메모리 컨텐츠들만이 커널의 권한으로 실행되는 것을 보장하기 위하여 이 기능을 채택할 수 있다.

[79] 페이지 테이블 속성들을 가지는 기존 보호는, 페이지 테이블들을 직접 손상시키는 커널 위업(exploits)에 취약할 수 있다. 커널이 각 프로세스에 대해 하나 이상의 페이지 테이블들을 유지 관리하고 반복적으로 수정해야 하기 때문에, 페이지 테이블들은 운영 체제 커널에 의하여 쓰기 가능한 것으로 유지되어야 할 수 있다. 그러므로, 커널 취약점을 악용하는 공격자는 종래 기술에서 제안된 바와 같이, 공격자의 코드를 포함하는 물리적 페이지를 포인팅하는 권한 있는 실행 가능한 가상 페이지를 생성하기 위하여 커널이 페이지 테이블 엔트리를 직접 손상시키도록 속일 수 있다.

[80] 따라서, 커널 코드 무결성은 페이지 테이블이 악의적인 수정들로부터 자유로운 경우에만 보장될 수 있다. 기존 메커니즘들은 종종 작은 임베디드 시스템들에서 바람직하지 않은 페이지 테이블을 손상시키는 이러한 공격들을 물리치기 위하여 다른 신뢰할 수 있는 소프트웨어 구성 요소에 의존할 수 있다. 예를 들어, 기존 메커니즘은, 페이지 테이블 업데이트를 ARM TruseZone에 의해 보호되는 신뢰할 수 있는 소프트웨어로 리디렉션할 수 있다. 다른 예를 들어, 기존 메커니즘은, 페이지 테이블 무결성을 위해 하이퍼바이저(hypervisor)에 의존할 수 있다.

[81] 본 명세서에서, 커널이 페이지 테이블 무결성에 의존하지 않고 커널 코드 무결성을 보호하는 것을 가능하게 하는 작은 하드웨어 확장으로서 일 실시예에 따른 프로세서(본 명세서에서, 'PRIVLOCK'라고도 표현됨)이 개시될 수 있다. 일 실시예에 따른 프로세서를 이용하여, 커널은 프로세서가 커널의 권한으로 커널 승인된 코드만 실행하는 것을 보장할 수 있다. 일 실시예에 따른 프로세서는 임베디드 시스템에서 커널 코드 페이지들이 관리되는 방법에 대한 세 가지 관찰에 기반할 수 있다.

[82] 첫째, 커널 코드를 포함하는 물리적 또는 가상 페이지들의 세트는 자주 변경되지 않을 수 있다. 코어 커널을 포함하는 대부분의 페이지들은, 부팅 시 할당(allocate)되고 런타임 동안 온전하게(intact) 유지될 수 있다. 운영 체제 커널들은 장치 드라이버들과 같은 추가 기능들을 위해 추가 코드를 동적으로 로딩하지만, 이는 특히 주변 장치들의 고정된 세트를 자주 이용하는 임베디드 시스템에서 드물게 발생할 수 있다.

[83] 둘째, 운영 체제는 물리적 주소 공간에서 메모리 페이지를 관리하는 데 어느 정도의 유연성을 가질 수 있다. 프로그램들은 코드 또는 데이터 메모리에 액세스하

기 위하여 가상 주소들을 이용하고, 커널은 필요한 경우 이러한 페이지들을 물리적 주소 공간에 다시 위치시킬 수 있다.

[84] 셋째, 메모리 관리 장치(memory management unit; MMU)는 커널 코드 페이지들을 보호하는 데 필요한 모든 속성 확인들을 구현할 수 있다. 유일한 추가 요구사항은 커널 코드 무결성을 보호하기 위하여 원하는 액세스 제어 정책을 준수하도록 속성들을 적절하게 제거(properly sanitize)하는 것을 포함할 수 있다.

[85] 이러한 관찰들 외에도, 일 실시예에 따른 프로세서는 승인된 커널 코드를 포함하는 주소 범위들 또는 페이지들의 세트를 정의하는 데 커널이 이용하는 추가 프로세서 제어 레지스터들(additional inprocessor control registers)(예: RISC-V 프로세서의 제어 및 상태 레지스터들(Control and Status Registers; CSRs))를 도입할 수 있다. 값을 이용하여, 일 실시예에 따른 프로세서는 프로세서가 특권 모드에서 실행되는 동안 지정된 물리적 메모리 페이지에서만 명령어들을 가져오고 페이지에 쓰기를 거부하는 것을 보장할 수 있다. 이 아키텍처 확장을 통해, 커널 코드 무결성은 더 이상 페이지 테이블의 무결성에 의존하지 않을 수 있다. 커널은 필요에 따라 커널의 페이지 테이블을 자유롭게 관리할 수 있다. 추가 제어 레지스터들의 무결성은, 많은 임베디드 시스템들에서 자주 해당하는, 시스템이 추가적인 코드를 로딩하지 않아야 하는 것을 가정하면, 시스템 부팅 후 일부 중요한 읽기 전용 레지스터를 쓰기 방지(write-protect)하는 데 일반적으로 사용되는 잠금 메커니즘에 의해 보호될 수 있다.

[86] 일 실시예에 따르면, 위협 모델은 커널 수준 공격에 대한 방어 메커니즘이 가정하는 일반적인 위협 모델을 따를 수 있다. 공격자는, 공격자가 지정한 메모리 위치들을 커널의 권한으로 읽거나 수정하도록 커널을 속일 수 있는 하나 이상의 커널 취약점들을 알고 있는 것으로 가정될 수 있다. 이는 커널이 정기적으로 읽고 수정하는 페이지 테이블을 포함할 수 있다. 운영 체제 커널 및 언더라잉 프로세서(underlying processor)는 이를 무력화하기 위해 최첨단 방어 메커니즘을 구현할 수 있다. 예를 들어, 페이지 테이블 엔트리는 PXN/SMEP와 같은 속성들을 포함하고, 커널은 특정 페이지를 실행 가능하지 않은 권한으로 만들 수 있다.

[87] 커널은 또한 합법적인 커널 코드 페이지에 대한 페이지 테이블 엔트리만 실행 가능한 권한으로 설정되도록 신중하게 작성될 수 있다. 이 위협 모델에서, 공격자는 페이지 테이블을 직접 수정함으로써 페이지에 대한 실행 가능한 권한 매핑을 생성하는 악성 코드 스니펫이 포함된 메모리 페이지를 준비함으로써 이 방어를 우회할 수 있다. 공격자는 그 이후에, 커널이 페이지로부터 실행되도록 할 수 있다. 일 실시예에 따른 프로세서는 커널 코드에 대해 구성된 물리적 페이지들을 이용하여 각 TLB 리필의 페이지 테이블 엔트리들을 소독(sanitizing)함으로써 이를 중지할 수 있다.

[88] 일 실시예에 따르면, 커널 코드 무결성을 효율적이고 효과적으로 보호하기 위해, 하드웨어는 두 번 변경되고, 운영 체제 커널은 한 번 변경될 수 있다. 그림 1은 두 가지 하드웨어 변경 사항이 나타날 수 있다. 일 실시예에 따른 프로세서는

커널 코드 페이지의 위치와 크기를 유지하는 추가 제어 레지스터(그림 2에서 후술됨), 및 페이지 테이블 워커에서 정책을 시행하기 위하여 페이지 테이블 엔트리를 검사하는 조합 로직을 추가할 수 있다. 커널에서, 페이지 폴트 핸들러(page fault handler)는 페이지 폴트(page fault)가 일 실시예에 따른 프로세서에 의하여 시행된 정책의 위반으로 인한 것인지 추가로 확인하도록 수정될 수 있다.

[89] 이하, RISC-V 프로세서와 Linux 커널을 대상으로 하는 설계 및 구현에 대해 자세히 설명한다. 다만, 이에 한정하는 것은 아니고, 일 실시예에 따른 메커니즘은 가상 메모리 시스템이 있는 모든 운영 체제 커널 및 범용 프로세서에 적용될 수 있다. 프로토타입에 대한 자세한 내용은 도 10 내지 도 13에서 후술한다.

[90] 도 1에서 나타난 바와 같이, 일 실시예에 따른 프로세서는 하드웨어 구성요소들을 포함할 수 있다. 동작(①)에서, 프로세서는, 부팅 시 커널 코드 범위들을 설정(configure)할 수 있다. 동작(②)에서, 프로세서는, 설정된 커널 코드 범위들에 기초하여, 페이지 테이블 워커(PTW)에 포함된 범위 레지스터들(range registers)을 설정할 수 있다.

[91] 그 이후에, 프로세서는 커널 모드인 동안, 액세스 요청을 수신할 수 있다. 프로세서는 커널 코드 영역 판단부(kernel code region determiner)(PRIVLOCKCHECKER)를 포함할 수 있다. 동작(③)에서, 프로세서의 커널 코드 영역 판단부(PRIVLOCKCHECKER)는, 액세스 요청의 가상 주소에 매핑된 물리 주소를 수신할 수 있다. 프로세서의 커널 코드 영역 판단부(PRIVLOCKCHECKER)는 범위 레지스터와 수신된 물리 주소와 비교함으로써, 물리 주소에 대응하는 영역(예: 물리 주소에 대응하는 메모리의 부분 영역)이 커널 코드 영역에 포함되는지 여부를 판단할 수 있다. 프로세서는, 물리 주소에 대응하는 영역에 대한 동작에 대한 인에이블 또는 디스에이블 여부를 결정할 수 있다. 프로세서는, 결정된 동작에 대한 인에이블 또는 디스에이블 여부에 기초하여 페이지 테이블 속성을 업데이트(예: 소독(sanitize))할 수 있다. 동작(④)에서, 프로세서의 커널 코드 영역 판단부(PRIVLOCKCHECKER)는, 변환 색인 버퍼(TLB)에게 업데이트된 페이지 테이블 속성을 가지는 페이지 테이블 엔트리(sanitized_pte)를 전달할 수 있다. 프로세서의 구체적인 동작은 도 2에서 후술한다.

[92] 도 2는 일 실시예에 따른 프로세서가 커널 코드 영역에 대한 동작을 제어하기 위한 방법을 설명하기 위한 도면이다.

[93] 일 실시예에 따른 커널 코드 영역에 대한 동작을 제어하기 위한 전자 장치는, 프로세서를 포함할 수 있다. 이하, 커널 코드 영역에 대한 동작을 제어하기 위한 프로세서의 동작을 설명한다.

[94] 단계(210)에서, 프로세서는 머신 모드(machine mode)인 동안, 범위 레지스터(range register)를 설정(configure)할 수 있다. 프로세서는, 커널 코드(kernel code)가 저장된 커널 코드 영역의 물리 주소(physical address)에 기초하여 범위 레지스터를 설정할 수 있다. 예를 들어, 프로세서는 부팅(booting) 시, 머신 모드일 수 있다.

프로세서는 운영 체제 커널을 머신 모드에서 실행하는 동안, CSRFile 인터페이스를 통해 코어(예: 로켓 코어(Rocket Core))의 범위 레지스터를 설정할 수 있다.

[95] 일 실시예에 따르면, 물리적 주소의 비트 개수를 가지는 컨트롤 레지스터는 도입될 수 있다. 예시적으로, 물리적 주소의 비트 개수는 32일 수 있다. 컨트롤 레지스터의 개수는 설계에 따라 설정될 수 있다. 예시적으로, 프로세서는 4개의 컨트롤 레지스터들을 포함할 수 있다. 범위 레지스터 및 범위 레지스터의 설정은 도 3 내지 도 4에서 후술한다.

[96] 단계(220)에서, 프로세서는 커널 모드(kernel mode)인 동안, 페이지(page)의 가상 주소에 기초한 액세스 요청을 수신하는 경우에 응답하여, 페이지의 물리 주소를 획득할 수 있다. 프로세서는, 페이지 테이블 엔트리(page table entry)를 이용하여, 액세스 요청의 가상 주소에 매핑된 물리 주소를 획득할 수 있다. 페이지 테이블 엔트리는, 액세스 요청의 가상 주소 및 해당 가상 주소와 매핑된 물리 주소에 관한 정보를 포함할 수 있다. 물리 주소의 획득은 도 5에서 후술한다.

[97] 단계(230)에서, 프로세서는, 설정된 범위 레지스터를 획득된 물리 주소와 비교함으로써, 획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는지 여부를 판단할 수 있다. 예를 들어, 프로세서의 페이지 테이블 워커(PTW)는 커널 코드 영역 판단부(예: 도 1의 커널 코드 영역 판단부(PRIVLOCKCHECKER))를 포함할 수 있다. 프로세서의 커널 코드 영역 판단부는, 액세스 요청의 가상 주소에 매핑된 물리 주소를 수신할 수 있다. 프로세서의 커널 코드 영역 판단부는 범위 레지스터와 수신된 물리 주소와 비교함으로써, 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는지 여부를 판단할 수 있다.

[98] 일 실시예에 따르면, 페이지 테이블 워커(PTW)는 복수의 범위 레지스터들을 포함할 수 있다. 커널 코드 영역 판단부는, 복수의 범위 레지스터들 각각에 대하여, 해당 범위 레지스터와 물리 주소를 비교함으로써, 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는지 여부를 판단할 수 있다. 커널 코드 영역 판단부는, 물리 주소에 대응하는 영역이 복수의 범위 레지스터들 중 적어도 하나의 범위 레지스터에 의하여 결정된 범위에 포함되는 것에 기초하여, 해당 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는 것을 결정할 수 있다. 커널 코드 영역 판단부는, 물리 주소에 대응하는 영역이 복수의 범위 레지스터들에 의하여 결정된 범위들에 포함되지 않는 것에 기초하여, 해당 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되지 않는 것을 결정할 수 있다. 범위 레지스터 및 물리 주소의 비교를 통한 물리 주소의 커널 코드 영역 포함 여부 판단은 도 6에서 후술한다.

[99] 단계(240)에서, 프로세서는 획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는지 여부에 기초하여, 물리 주소에 대응하는 영역에 대한 동작에 대한 인에이블(enable) 또는 디스에이블(disable) 여부를 결정할 수 있다. 물리 주소에 대응하는 영역에 대한 동작은 쓰기 동작 또는 실행 동작 중 적어도 하나의 동작을 포함할 수 있다.

[100] 메모리의 영역에 대한 쓰기 동작은, 메모리의 영역에 데이터를 저장하는 동작으로서, 해당 영역에 커널 모드인 동안 데이터를 저장하는 커널 모드의 쓰기 동작 및 해당 영역에 사용자 모드인 동안 데이터를 저장하는 사용자 모드의 쓰기 동작을 포함할 수 있다.

[101] 메모리의 영역에 대한 실행 동작은, 메모리의 영역에 저장된 데이터(예: 프로그램 코드)에 기초한 프로그램을 실행하는 동작으로서, 커널 모드인 동안 해당 영역에 저장된 데이터에 기초한 프로그램을 실행하는 커널 모드의 실행 동작 및 사용자 모드인 동안 해당 영역에 저장된 데이터에 기초한 프로그램을 실행하는 사용자 모드의 실행 동작을 포함할 수 있다. 동작의 인에이블 또는 디스에이블 여부의 결정은 도 7에서 후술한다.

[102] 일 실시예에 따르면, 페이지 테이블 엔트리는 가상 주소 및 해당 가상 주소에 매핑된 물리 주소를 포함할 수 있다. 페이지 테이블 엔트리는, 페이지 테이블 속성과 함께 저장될 수 있다. 페이지 테이블 속성은, 대응하는 페이지 테이블 엔트리의 물리 주소를 가지는 메모리 영역에 대한 동작(예: 커널 모드의 쓰기 동작, 사용자 모드의 쓰기 동작, 커널 모드의 쓰기 동작, 사용자 모드의 실행 동작)의 인에이블 또는 디스에이블 여부에 관한 정보를 포함할 수 있다. 예를 들어, 페이지 테이블 속성은, 커널 모드의 쓰기 동작의 인에이블 여부를 지시하는 비트(예: sw 비트), 사용자 모드의 쓰기 동작의 인에이블 여부를 지시하는 비트(예: w 비트), 커널 모드의 실행 동작의 인에이블 여부를 지시하는 비트(예: sx 비트), 및 사용자 모드의 실행 동작의 인에이블 여부를 지시하는 비트(예: x 비트)를 가질 수 있다. 페이지 테이블 속성의 업데이트는 도 8에서 후술한다.

[103] 도 3은 일 실시예에 따른 범위 레지스터의 설정 동작을 설명하기 위한 도면이다.

[104] 도 4는 일 실시예에 따른 복수의 비트들을 가지는 범위 레지스터의 구조를 설명하기 위한 도면이다.

[105] 단계(310)에서, 프로세서는, 머신 모드인 동안, 커널 코드를 가지는 페이지를 미리 결정된 크기를 가지는 청크(chunk)의 경계(boundary)에 따라 정렬(align)할 수 있다.

[106] 후술하겠으나, 프로세서는 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는지 여부를 판단하기 위하여, 복수의 비트들로 표현된 물리 주소의 일부를 범위 레지스터와 비교할 수 있다. 예시적으로, 프로세서는 물리 주소 중 하나 이상의 상위 비트들(upper bits)에 대응하는 부분 주소를 범위 레지스터와 비교할 수 있다. 프로세서는, 물리 주소에 대응하는 메모리 영역이 커널 코드 영역에 포함되는지 여부를 판단하는 데 물리 주소 중 상위 비트들에 대응하는 부분 주소가 요구되도록, 커널 코드를 저장할 수 있다. 프로세서는 커널 코드가 저장된 커널 코드 영역을 미리 결정된 크기를 가지는 청크(예: 16MB의 청크, 16KB의 청크)의 경계에 따라 정렬할 수 있다. 예시적으로, 16MB는 2^{24}에 대응하므로, 16MB의 청크의 경계는, 32 비트들로 표현된 물리 주소 중 8개의 상위 비트들에 의하여 결정

될 수 있다. 16MB의 청크의 경계는 24개의 하위 비트들 각각이 모두 0의 값을 가질 수 있다.

[107] 단계(320)에서, 프로세서는, 커널 코드를 적어도 하나의 커널 코드 청크(kernel code chunk)로 저장할 수 있다. 커널 코드 청크는, 커널 코드의 적어도 일부가 연속적으로 저장된 메모리의 영역을 의미할 수 있다. 커널 코드 청크는, 커널 코드에 대응하는 커널 코드 페이지로 지칭될 수 있다. 커널 코드는 복수의 커널 코드 청크들로 저장될 수 있다.

[108] 일 실시예에 따르면, 커널 코드 영역 판단부(예: 도 1의 커널 코드 영역 판단부(PRIVLOCKCHECKER))는 각 커널 코드 청크가 크기에 맞춰 정렬되어야 하는 것을 요구하기 때문에, 커널 코드 페이지는 일 실시예에 따른 프로세서에 의해 보호되고 정렬되어야 할 수 있다. 성능에 영향을 미치지 않으면서 작은 변경으로 커널 코드 청크를 정렬할 수 있는 것이 발견될 수 있다. 운영 체제 커널들은 일반적으로 수 메가바이트에서 수십 메가바이트의 코드를 가질 수 있다. 예를 들어, 구현에서 실행되는 Linux 커널은 약 3.04MB의 코드를 가질 수 있다. 커널 코드 청크가 2MB 경계에 맞춰질 수 있다. 결과적으로, 2개의 2MB의 커널 코드 청크들은 초래될 수 있고, 프로세서는 2개의 2MB의 커널 코드 청크들에 대한 2개의 범위 레지스터들의 사용할 수 있다. 커널 코드 청크의 정렬은 링커 스크립트(linker script)가 변경됨으로써 수행될 수 있다.

[109] 단계(330)에서, 프로세서는, 각 커널 코드 청크에 대하여, 해당 커널 코드 청크에 대응하는 범위 레지스터의 적어도 일부를 설정할 수 있다. 프로세서는, 복수의 커널 코드 청크들을 통해 커널 코드를 저장한 경우, 복수의 커널 코드 청크들에 대응하는 복수의 범위 레지스터들을 포함할 수 있다. 복수의 범위 레지스터들 각각은, 복수의 커널 코드 청크들 중 하나의 커널 코드 청크에 대응할 수 있다.

[110] 일 실시예에 따르면, 프로세서는, 각 커널 코드 청크의 시작 물리 주소에 기초하여, 해당 커널 코드 청크에 대응하는 범위 레지스터의 베이스 비트(base bit)를 설정할 수 있다. 범위 레지스터의 베이스 비트는, 범위 레지스터가 대응하는 커널 코드 청크의 시작 물리 주소에 관한 정보를 가질 수 있다. 예시적으로, 범위 레지스터의 복수의 비트들 중 20개의 상위 비트들인 베이스 비트는, 범위 레지스터가 대응하는 커널 코드 청크의 시작 물리 주소에 기초하여 결정될 수 있다. 예를 들어, 프로세서는 베이스 비트 중 일부(예: 18개의 비트들)는, 범위 레지스터가 대응하는 커널 코드 청크의 시작 물리 주소와 같은 값으로 결정될 수 있다.

[111] 예시적으로, 범위 레지스터는 32개의 비트들을 가질 수 있다. 베이스 비트는 범위 레지스터의 상위 20개 비트들일 수 있다. 20개의 비트들을 가지는 베이스 비트는, 2개의 상위 비트들은 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는지 여부의 판단과 독립적인 범위 레지스터에 대한 정보에 관한 비트들일 수 있다. 베이스 비트 중 2개의 상위 비트들 외의 18개의 비트들은, 범위 레지스터가 대응하는 커널 코드 청크의 시작 물리 주소와 같은 값을 가질 수 있다.

[112] 일 실시예에 따르면, 프로세서는, 각 커널 코드 청크의 크기에 기초하여, 해당 커널 코드 청크에 대응하는 범위 레지스터의 마스크 비트를 설정할 수 있다. 범위 레지스터의 마스크 비트는, 범위 레지스터가 대응하는 커널 코드 청크의 크기에 관한 정보를 가질 수 있다.

[113] 예시적으로, 마스크 비트는 베이스 비트에 후속하는 범위 레지스터의 10개 비트들일 수 있다. 10개의 비트들을 가지는 마스크 비트는, 각 커널 코드 청크의 크기에 기초하여 설정될 수 있다. 후술하겠으나, 프로세서는 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는지 여부를 판단하기 위하여, 물리 주소의 일부와 범위 레지스터의 적어도 일부를 비교할 수 있다. 범위 레지스터의 적어도 일부(예: 베이스 비트)와 비교할 물리 주소의 상위 비트들은 커널 코드 청크의 크기에 기초하여 결정될 수 있다.

[114] 이하, 물리 주소는 32개의 비트들로 표현되는 경우 커널 코드 청크의 크기에 따라 마스크 비트를 설정하는 동작의 예시에 관하여 설명한다.

[115] 예시적으로, 커널 코드 청크의 크기가 16MB인 경우, 16MB는 2^{24}에 대응하므로, 해당 커널 코드 청크에 포함된 물리 주소의 24개의 하위 비트들은 서로 다른 값들을 가질 수 있다. 다시 말해, 해당 커널 코드 청크에 포함되었는지 여부는 물리 주소의 상위 8개의 비트들에 기초하여 결정될 수 있다. 프로세서는 범위 레지스터와 비교할 물리 주소의 상위 비트들을 8개의 상위 비트들로 결정할 수 있다. 프로세서는, 범위 레지스터와 비교할 물리 주소의 일부를, 미리 결정된 개수(예: 8개)에 마스크 비트에 기초하여 결정된 개수가 합산된 개수의 상위 비트들로 결정할 수 있다. 프로세서는, 커널 코드 청크의 크기가 16MB인 경우, 마스크 비트를 $0000000000_{(2)}$으로 설정할 수 있다. 프로세서는, 마스크 비트가 $0000000000_{(2)}$으로 설정된 경우, 물리 주소 중에서, 미리 결정된 개수(예: 8개)에 마스크 비트에 기초하여 결정된 개수(예: 0개)가 합산된 개수(예: 8개)의 상위 비트들을 범위 레지스터와 비교할 수 있다.

[116] 예시적으로, 커널 코드 청크의 크기가 1MB인 경우, 1MB는 2^{20}에 대응하므로, 해당 커널 코드 청크에 포함된 물리 주소의 20개의 하위 비트들은 서로 다른 값들을 가질 수 있다. 다시 말해, 해당 커널 코드 청크에 포함되었는지 여부는 물리 주소의 상위 12개의 비트들에 기초하여 결정될 수 있다. 프로세서는 범위 레지스터와 비교할 물리 주소의 상위 비트들을 12개의 상위 비트들로 결정할 수 있다. 프로세서는, 범위 레지스터와 비교할 물리 주소의 일부를, 미리 결정된 개수(예: 8개)에 마스크 비트에 기초하여 결정된 개수가 합산된 개수의 상위 비트들로 결정할 수 있다. 프로세서는, 커널 코드 청크의 크기가 1MB인 경우, 마스크 비트를 $1111000000_{(2)}$으로 설정할 수 있다. 프로세서는, 마스크 비트가 $1111000000_{(2)}$으로 설정된 경우, 물리 주소 중에서, 미리 결정된 개수(예: 8개)에 마스크 비트에 기초하여 결정된 개수(예: 4개)가 합산된 개수(예: 12개)의 상위 비트들을 범위 레지스터와 비교할 수 있다.

[117] 예시적으로, 커널 코드 청크의 크기가 16KB인 경우, 16KB는 2^{14}에 대응하므로, 해당 커널 코드 청크에 포함된 물리 주소의 14개의 하위 비트들은 서로 다른 값들을 가질 수 있다. 다시 말해, 해당 커널 코드 청크에 포함되었는지 여부는 물리 주소의 상위 18개의 비트들에 기초하여 결정될 수 있다. 프로세서는 범위 레지스터와 비교할 물리 주소의 상위 비트들을 18개의 상위 비트들로 결정할 수 있다. 프로세서는, 범위 레지스터와 비교할 물리 주소의 일부를, 미리 결정된 개수(예: 18개)에 마스크 비트에 기초하여 결정된 개수가 합산된 개수의 상위 비트들로 결정할 수 있다. 프로세서는, 커널 코드 청크의 크기가 16MB인 경우, 마스크 비트를 $1111111111_{(2)}$으로 설정할 수 있다. 프로세서는, 마스크 비트가 $1111111111_{(2)}$으로 설정된 경우, 물리 주소 중에서, 미리 결정된 개수(예: 8개)에 마스크 비트에 기초하여 결정된 개수(예: 10개)가 합산된 개수(예: 18개)의 상위 비트들을 범위 레지스터와 비교할 수 있다.

[118] 일 실시예에 따르면, 프로세서는, 범위 레지스터의 베이스 비트 및 마스크 비트를 설정한 경우에 응답하여, 범위 레지스터의 밸리드 비트(valid bit) 및 락 비트(lock bit)를 설정할 수 있다.

[119] 범위 레지스터의 밸리드 비트는, 범위 레지스터가 커널 코드 청크에 대응하는지 여부에 관한 정보를 가질 수 있다. 범위 레지스터들의 개수가 커널 코드 청크의 개수보다 많은 경우, 범위 레지스터들 중 적어도 하나의 범위 레지스터는 커널 코드 청크에 대응하지 않을 수 있다. 예시적으로, 범위 레지스터들이 4개이고, 커널 코드 청크가 3개인 경우, 범위 레지스터들 중 3개의 범위 레지스터들 각각은 하나의 커널 코드 청크에 대응하고, 범위 레지스터들 중 나머지 1개의 범위 레지스터는 커널 코드 청크에 대응하지 않을 수 있다.

[120] 예시적으로, 밸리드 비트는 마스크 비트에 후속하는 1개의 비트일 수 있다. 프로세서는, 커널 코드 청크에 기초하여 범위 레지스터의 베이스 비트, 및 마스크 비트를 설정한 경우에 응답하여, 밸리드 비트의 값을 1로 설정할 수 있다. 프로세서는 커널 코드 청크가 대응하는 범위 레지스터를 설정한 이후, 설정되지 않은 범위 레지스터에 대하여 밸리드 비트의 값을 0으로 설정할 수 있다.

[121] 프로세서는 범위 레지스터의 밸리드 비트가 1로 설정된 경우, 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는지 여부의 판단에 해당 범위 레지스터를 이용할 수 있다. 범위 레지스터의 밸리드 비트가 0으로 설정된 경우, 프로세서는 해당 범위 레지스터는 커널 코드 청크에 대응하지 않는 것으로 판단할 수 있다. 결과적으로, 프로세서는, 범위 레지스터의 밸리드 비트가 0으로 설정된 경우, 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는지 여부의 판단으로부터 해당 범위 레지스터의 이용을 배제할 수 있다.

[122] 범위 레지스터의 락 비트는, 범위 레지스터의 설정 완료 여부에 관한 정보를 가질 수 있다. 락 비트는, 한 번 설정되는 비울 수 없는 스티키(sticky) 비트일 수 있다. 예시적으로, 락 비트는 밸리드 비트에 후속하는 1개의 비트일 수 있다. 프로세서는, 머신 모드인 동안, 커널 코드 청크에 기초하여 범위 레지스터의 베이스

비트, 마스크 비트, 및 밸리드 비트를 설정할 수 있다. 프로세서는, 커널 코드 청크에 기초하여 범위 레지스터의 베이스 비트, 마스크 비트, 및 밸리드 비트를 설정한 이후에, 해당 범위 레지스터의 락 비트를 1로 설정할 수 있다. 프로세서는, 범위 레지스터의 락 비트이 1로 설정된 경우, 범위 레지스터에 대한 추가적인 쓰기 동작 및/또는 업데이트 동작을 무시할 수 있다. 프로세서는, 커널 코드 청크가 대응하는 범위 레지스터를 설정한 이후에, 나머지 범위 레지스터에 대하여 밸리드 비트의 값을 0으로 설정한 이후에, 0으로 설정된 밸리드 비트의 범위 레지스터의 락 비트를 1로 설정할 수 있다.

[123] 일 실시예에 따른 프로세서는, 범위 레지스터의 락 비트를 통해, 범위 레지스터를 업데이트하기 위한 특별한 명령어들을 실행 가능한 강력한 공격자(attacker)일지라도 이미 설정된 범위 레지스터들을 손상시킬 수 없으므로, 범위 레지스터들에 의하여 결정된 범위들을 변경할 수 없다. 따라서, 일 실시예에 따른 프로세서는 범위 레지스터의 락 비트의 값이 1로 설정된 이후, 해당 범위 레지스터에 대한 추가적인 쓰기 동작 및/또는 업데이트 동작을 무시할 수 있다.

[124] 도 5는 일 실시예에 따른 프로세서가 액세스 요청의 가상 주소에 매핑된 물리 주소를 획득하는 동작을 설명하기 위한 도면이다.

[125] 일 실시예에 따른 프로세서는, 커널 모드인 동안 가상 주소에 액세스 요청을 수신한 경우에 응답하여, 가상 주소에 대한 페이지 테이블 엔트리를 이용하여 가상 주소에 매핑된 물리 주소를 획득할 수 있다. 가상 주소에 대한 페이지 테이블 엔트리는, 변환 색인 버퍼(translation lookaside buffer; TLB)에 캐시(cache)되어 있을 수 있고, 또는 변환 색인 버퍼에 캐시되어 있지 않을 수도 있다.

[126] 단계(510)에서, 프로세서는 변환 색인 버퍼에 액세스 요청의 가상 주소에 대한 페이지 테이블 엔트리가 캐시되어 있는지 여부를 확인할 수 있다. 일 실시예에 따르면, 최근에 이용된 페이지 테이블 엔트리는 변환 색인 버퍼에 캐시되어 있을 수 있다.

[127] 도 8에서 후술하겠으나, 일 실시예에 따르면, 프로세서는 페이지 테이블 엔트리는 변환 색인 버퍼에 캐시되는 경우, 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는지 여부에 기초하여, 페이지 테이블 엔트리에 대한 페이지 테이블 속성을 업데이트할 수 있다. 프로세서는 업데이트된 페이지 테이블 속성과 함께 페이지 테이블 엔트리를 변환 색인 버퍼에 캐시할 수 있다. 따라서, 일 실시예에 따르면, 프로세서는, 변환 색인 버퍼에 이미 캐시되어 있는 페이지 테이블 엔트리의 페이지 테이블 속성이 커널 코드 영역 판단부의 판단에 기초하여 업데이트된 것으로 취급할 수 있다. 예를 들어, 프로세서는, 변환 색인 버퍼에 캐시된 페이지 테이블 엔트리의 페이지 테이블 속성은 공격자에 의하여 조작되지 않은 것으로 취급할 수 있다.

[128] 단계(520)에서, 프로세서는 변환 색인 버퍼에 페이지 테이블 엔트리가 캐시되어 있지 않은 경우에 응답하여, 페이지 테이블로부터 가상 주소에 대한 페이지 테이블 엔트리를 검색할 수 있다. 페이지 테이블 엔트리는, 가상 주소 및 가상 주

소에 매핑된 물리 주소를 가질 수 있다. 프로세서는, 액세스 요청의 가상 주소에 대한 페이지 테이블 엔트리가 변환 색인 버퍼에 캐시되어 있지 않은 것을 확인할 수 있다. 프로세서는, 액세스 요청의 가상 주소에 기초하여, 페이지 테이블로부터 가상 주소에 대한 페이지 테이블 엔트리를 검색할 수 있다.

[129] 단계(530)에서, 프로세서는 검색된 페이지 테이블 엔트리에 기초하여, 가상 주소에 매핑된 페이지의 물리 주소를 획득할 수 있다.

[130] 단계(540)에서, 프로세서는 변환 색인 버퍼에 페이지 테이블 엔트리가 캐시되어 있는 경우에 응답하여, 변환 색인 버퍼에 캐시된 페이지 테이블 엔트리에 기초하여 페이지의 물리 주소를 획득할 수 있다.

[131] 도 6는 일 실시예에 따른 프로세서가 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는지 여부를 판단하는 동작을 설명하기 위한 도면이다.

[132] 단계(610)에서, 프로세서는 범위 레지스터의 마스크 비트에 기초하여, 물리 주소 중 부분 주소를 결정할 수 있다. 부분 주소는, 물리 주소 중에서 범위 레지스터와 비교할 부분 주소를 의미할 수 있다. 예를 들어, 물리 주소는 32개의 비트들로 표현될 수 있다. 범위 레지스터의 마스크 비트가 $1111000000_{(2)}$의 값으로 설정된 경우, 프로세서는 물리 주소 중에서, 미리 결정된 개수(예: 8개)에 마스크 비트에 의하여 결정된 개수(예: 4개)가 합산된 개수(예: 12개)의 상위 비트들을 부분 주소로 결정할 수 있다.

[133] 단계(620)에서, 프로세서는, 결정된 부분 주소가 범위 레지스터의 베이스 비트의 적어도 일부에 매칭되는 경우에 응답하여, 획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는 것을 판단할 수 있다. 프로세서는 물리 주소에 저장된 코드가 커널 코드인 것을 판단할 수 있다.

[134] 예시적으로, 범위 레지스터의 베이스 비트의 값은 14 비트들만큼 왼쪽으로 시프팅된 후에 커널 코드 청크의 시작 물리 주소로서 이용될 수 있다. 범위 레지스터의 마스크 비트의 값은 14비트들만큼 왼쪽으로 시프팅될 수 있고, 마스크 값을 구성하기 위해 마스크 비트의 값에 선행하는 미리 결정된 개수의 비트들이 1로 설정되어 추가(prepend)될 수 있다. 예시적으로, 프로세서는 물리 주소(addr)에 대응하는 영역은 수학식 1에 따라 커널 코드 영역에 포함되는지 여부를 판단할 수 있다.

[135] [수학식 1]

[136] $(BASE \ll 14) == (addr \,\&\, ((0\chi FF \ll 24) | (MASK \ll 14)))$

[137] 프로세서는, 물리 주소(addr)가 수학식 1을 만족하는 경우, 물리 주소에 대응하는 영역이 범위 레지스터가 대응하는 커널 코드 영역에 포함되는 것을 결정할 수 있다. 프로세서는, 물리 주소(addr)가 수학식 1을 만족하지 않는 경우, 물리 주소에 대응하는 영역이 범위 레지스터가 대응하는 커널 코드 영역에 포함되지 않는 것을 결정할 수 있다.

[138] 도 7는 일 실시예에 따른 프로세서가 물리 주소에 대응하는 영역에 대한 동작의 인에이블 또는 디스에이블 여부를 결정하는 동작을 설명하기 위한 도면이다.

[139] 단계(710)에서, 프로세서는 획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는 경우에 응답하여, 물리 주소에 대응하는 영역에 대한 쓰기 동작(write operation)을 디스에이블할 수 있다.

[140] 예를 들어, 프로세서는, 물리 주소에 대응하는 영역에 대한 커널 모드의 쓰기 동작을 디스에이블할 수 있다. 프로세서는 페이지 테이블 엔트리의 페이지 테이블 속성의 sw 비트를 0으로 설정함으로써, 커널 모드인 동안 물리 주소에 대응하는 영역에 대한 쓰기 동작을 제한할 수 있다.

[141] 예를 들어, 프로세서는, 물리 주소에 대응하는 영역에 대한 사용자 모드의 쓰기 동작을 디스에이블할 수 있다. 프로세서는 페이지 테이블 엔트리의 페이지 테이블 속성의 w 비트를 0으로 설정함으로써, 사용자 모드인 동안 물리 주소에 대응하는 영역에 대한 쓰기 동작을 제한할 수 있다.

[142] 비교 실시예에 따르면, 커널 코드 영역에 대한 쓰기 동작이 인에이블되는 경우, 공격자는 커널 코드 영역에 악성 코드를 저장할 수 있다. 커널 코드 영역에 악성 코드가 저장되면, 프로세서가 커널 모드인 동안 커널 코드 영역에 저장된 악성 코드를 실행할 수도 있다. 이와 달리, 일 실시예에 따르면, 프로세서는 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는 경우, 물리 주소에 대응하는 영역에 대한 쓰기 동작을 디스에이블함으로써, 공격자에 의하여 커널 코드 영역에 커널 코드와 다른 코드(예: 악성 코드)가 저장되는 것을 방지할 수 있다.

[143] 단계(720)에서, 프로세서는 획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는 경우에 응답하여, 커널 모드인 동안 물리 주소에 대응하는 영역에 대한 실행 동작(execute operation)을 인에이블할 수 있다.

[144] 예를 들어, 프로세서는, 물리 주소에 대응하는 영역에 대한 커널 모드의 실행 동작을 인에이블할 수 있다. 프로세서는 페이지 테이블 엔트리의 페이지 테이블 속성의 sx 비트를 1로 설정함으로써, 커널 모드인 동안 물리 주소에 대응하는 영역에 대한 실행 동작을 허용할 수 있다.

[145] 단계(730)에서, 프로세서는 획득된 물리 주소에 대응하는 영역이 커널 코드 영역과 다른 영역에 포함되는 경우에 응답하여, 커널 모드인 동안 물리 주소에 대응하는 영역에 대한 실행 동작을 디스에이블할 수 있다.

[146] 예를 들어, 프로세서는, 물리 주소에 대응하는 영역에 대한 커널 모드의 실행 동작을 디스에이블할 수 있다. 프로세서는 페이지 테이블 엔트리의 페이지 테이블 속성의 x 비트를 0으로 설정함으로써, 커널 모드인 동안 물리 주소에 대응하는 영역에 대한 실행 동작을 제한할 수 있다.

[147] 비교 실시예에 따르면, 공격자는 커널 코드와 다른 코드(예: 악성 코드)를 메모리의 일부 영역에 저장할 수 있다. 공격자는, 다른 코드가 저장된 일부 영역에 대한 커널 모드의 실행 동작을 인에이블할 수 있다. 다시 말해, 공격자는, 다른 코드를 메모리의 일부 영역에 저장하고, 프로세서가 메모리의 일부 영역을 커널 코드

영역으로 인식하도록 조작할 수 있다. 비교 실시예에 따른 프로세서는, 커널 모드인 동안 일부 영역에 저장된 악성 코드를 실행할 수도 있다. 이와 달리, 일 실시예에 따르면, 프로세서는 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되지 않는 경우, 물리 주소에 대응하는 영역에 대한 커널 모드의 실행 동작을 디스에이블함으로써, 커널 모드인 동안 다른 코드(예: 악성 코드)를 실행하는 것을 방지할 수 있다.

[148] 단계(740)에서, 프로세서는 페이지 테이블 엔트리의 페이지 테이블 속성과 독립적으로, 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정할 수 있다. 가상 주소에 대한 페이지 테이블 엔트리는, 페이지 테이블 속성을 가질 수 있다. 전술한 바와 같이, 페이지 테이블 속성은, 페이지 테이블 엔트리의 물리 주소에 대응하는 영역에 대하여 커널 모드의 쓰기 동작, 사용자 모드의 쓰기 동작, 커널 모드의 실행 동작, 또는 사용자 모드의 실행 동작 중 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부에 관한 정보를 포함할 수 있다.

[149] 일 실시예에 따른 프로세서는, 공격자에 의한 페이지 테이블 속성의 조작으로 인하여, 물리 주소에 대응하는 영역에 인에이블되어야 하는 동작을 디스에이블한 것으로 또는 디스에이블되어야 하는 동작을 인에이블한 것으로 속는 것을 방지할 수 있다.

[150] 예를 들어, 공격자는 커널 코드가 저장된 커널 코드 영역에 대하여 쓰기 동작이 디스에이블되어야 함에도 불구하고, 인에이블된 것으로 조작할 수 있다. 공격자는 커널 코드 영역에 대한 페이지 테이블 엔트리의 페이지 테이블 속성에서, w 비트 또는 sw 비트 중 적어도 하나의 비트를 1로 조작할 수 있다. 예를 들어, 공격자는 커널 코드 영역과 다른 영역에 대하여 실행 동작이 디스에이블되어야 함에도 불구하고, 인에이블된 것으로 조작할 수 있다. 예시적으로, 공격자는 커널 코드 영역과 다른 영역에 대한 페이지 테이블 엔트리의 페이지 테이블 속성에서, sx 비트를 1로 조작할 수 있다.

[151] 비교 실시예에 따르면, 프로세서는, 물리 주소에 대응하는 영역에 대한 쓰기 동작 및/또는 실행 동작의 인에이블 또는 디스에이블 여부를 페이지 테이블 속성에 기초하여 결정할 수 있다. 결과적으로, 공격자가 페이지 테이블 속성을 조작한 경우, 비교 실시예에 따른 프로세서는 조작된 페이지 테이블 속성에 기초하여 쓰기 동작 및/또는 실행 동작의 인에이블 또는 디스에이블 여부를 결정할 수 있다. 이와 달리, 일 실시예에 따른 프로세서는, 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는지 여부를 판단하고, 판단 결과에 기초하여 물리 주소에 대응하는 영역에 대한 쓰기 동작 및/또는 실행 동작의 인에이블 또는 디스에이블 여부를 결정할 수 있다. 결과적으로, 공격자가 페이지 테이블 속성을 조작한 경우에도, 일 실시예에 따른 프로세서는, 커널 코드 영역에 포함 여부의 판단 결과에 따라 올바른 동작의 인에이블 또는 디스에이블 여부를 결정할 수 잇다.

[152] 도 8은 일 실시예에 따른 프로세서가 액세스 요청의 가상 주소에 대한 페이지 테이블 엔트리가 변환 색인 버퍼에 캐시되어 있지 않은 경우 페이지 테이블 속성을 업데이트하는 동작을 설명하기 위한 도면이다.

[153] 일 실시예에 따르면, 전술한 바와 같이, 프로세서는 페이지 테이블 엔트리의 페이지 테이블 속성과 독립적으로, 물리 주소에 대응하는 영역에 대한 동작(예: 쓰기 동작, 실행 동작)의 인에이블 또는 디스에이블 여부를 결정할 수 있다. 프로세서는, 결정된 동작의 인에이블 또는 디스에이블 여부에 따라 페이지 테이블 속성을 설정할 수 있다.

[154] 단계(810)에서, 프로세서는 결정된 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부에 기초하여, 페이지 테이블 엔트리의 페이지 테이블 속성을 업데이트할 수 있다. 본 명세서에서, 페이지 테이블 엔트리의 페이지 테이블 속성을 업데이트하는 것은, 페이지 테이블 속성(또는 페이지 테이블 엔트리)을 소독(sanitize)하는 것으로도 표현될 수 있다.

[155] 예를 들어, 프로세서는 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는 것을 결정할 수 있다. 프로세서는, 물리 주소에 대응하는 영역에 대한 커널 모드의 쓰기 동작 및 사용자 모드의 쓰기 동작을 디스에이블할 수 있다. 프로세서는, 물리 주소에 대응하는 영역에 대한 커널 모드의 실행 동작을 인에이블할 수 있다. 프로세서는, 페이지 테이블 속성의 sw 비트 및 w 비트를 0으로 업데이트할 수 있다. 프로세서는, 페이지 테이블 속성의 sx 비트를 1로 업데이트할 수 있다.

[156] 예를 들어, 프로세서는 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되지 않는 것을 결정할 수 있다. 프로세서는, 물리 주소에 대응하는 영역에 대한 커널 모드의 실행 동작을 디스에이블할 수 있다. 프로세서는, 페이지 테이블 속성의 sw 비트 및 w 비트를 1로 업데이트할 수 있다. 프로세서는, 페이지 테이블 속성의 sx 비트를 0으로 업데이트할 수 있다.

[157] 일 실시예에 따르면, 프로세서는 공격자가 물리 주소에 대응하는 영역의 페이지 테이블 속성을 조작한 경우에도, 페이지 테이블 속성과 독립적으로 결정된 동작의 인에이블 또는 디스에이블 여부에 기초하여 페이지 테이블 속성을 업데이트함으로써, 페이지 테이블 속성을 참 값으로 유지할 수 있다.

[158] 단계(820)에서, 프로세서는 업데이트된 페이지 테이블 속성을 페이지 테이블 엔트리와 함께 변환 색인 버퍼에 캐시할 수 있다. 일 실시예에 따르면, 프로세서는, 최근에 검색된 페이지 테이블 엔트리를 변환 색인 버퍼에 캐시하는 정책을 이용할 수 있다. 이후, 프로세서는 변환 색인 버퍼에 캐시되어 있는 페이지 테이블 엔트리의 페이지 테이블 속성은 프로세서(예: 커널 코드 영역 판단부)에 의하여 업데이트된 페이지 테이블 속성일 수 있다.

[159] 도 9은 일 실시예에 따른 프로세서가 액세스 요청의 가상 주소에 대한 페이지 테이블 엔트리가 변환 색인 버퍼에 캐시되어 있는 경우 페이지 테이블 속성을 업데이트하는 동작을 설명하기 위한 도면이다.

[160] 일 실시예에 따르면, 전술한 바와 같이, 프로세서는 프로세서(예: 커널 코드 영역 판단부)에 의하여 업데이트된 페이지 테이블 속성을 변환 색인 버퍼에 캐시할 수 있다. 결과적으로, 프로세서는, 변환 색인 버퍼에 캐시되어 있는 페이지 테이블 엔트리의 페이지 테이블 속성에 대하여, 커널 코드 영역의 포함 여부 판단 및/또는 동작에 대한 인에이블 또는 디스에이블 여부의 결정을 수행하지 않더라도, 해당 페이지 테이블 속성은 공격자의 조작되지 않은 것으로 취급할 수 있다.

[161] 단계(910)에서, 프로세서는 변환 색인 버퍼에 페이지 테이블 엔트리가 캐시되어 있는 경우에 응답하여, 설정된 범위 레지스터와 획득된 물리 주소의 비교를 생략할 수 있다.

[162] 단계(920)에서, 프로세서는 페이지 테이블 엔트리의 페이지 테이블 속성에 기초하여 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정할 수 있다.

[163] 일 실시예에 따르면, 프로세서는 변환 색인 버퍼에 페이지 테이블 엔트리가 캐시되어 있는 경우, 페이지 테이블 속성은 공격자의 조작되지 않은 것으로 취급할 수 있다. 프로세서는, 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는지 여부를 판단하지 않더라도, 신뢰할 수 있는 페이지 테이블 속성에 포함된 동작의 인에이블 또는 디스에이블 여부를 결정할 수 있다.

[164] 일 실시예에 따른 프로세서는, 메모리에 대한 액세스 요청에 대하여, 동작의 인에이블 또는 디스에이블 여부를 주소 변환 후에 또는 주소 변환과 함께 검사할 수 있다. 다시 말해, 일 실시예에 따른 프로세서는, 페이지 테이블을 사용하여 주소를 변환하는 가상 주소 기반 메커니즘과 유사하게, 프로세서가 메모리에 액세스하는 데 사용하는 정확한 물리 주소에 기초하여 동작의 인에이블 또는 디스에이블 여부를 결정할 수 있다. 그렇지 않은 경우, 공격자는 특정 페이지 테이블 엔트리를 조작하여 검사를 우회하고 커널 코드 페이지를 실행하거나 손상시킬 수 있다.

[165] 일 실시예에 따른 프로세서는, 메모리에 대한 액세스 요청에 대하여, 캐시 액세스 전에 검사할 수 있다. 일 실시예에 따른 프로세서는 커널 코드 무결성을 보호하는 것을 목표로 하며, 커널 코드의 무결성을 위하여, 커널 코드를 저장하는 커널 코드 영역에 대한 쓰기 동작의 제한뿐만 아니라, 커널 모드인 동안 커널 코드 영역과 다른 영역에 저장된 다른 코드(예: 외부 코드)에 대한 실행 동작도 함께 제한되어야 할 수 있다. 프로세서는 일반적으로 커널 모드에 들어갈 때 캐시를 플러시하지 않으므로, 캐시는 사용자 프로그램에 대한 일부 코드가 포함될 수 있다. 추가 보안 메커니즘이 캐시 외부의 메모리 액세스를 검사하는 경우, 공격자는 커널이 캐시된 사용자 프로그램으로 점프하도록 함으로써 메커니즘을 우회할 수 있다. 이러한 코드가 커널의 권한으로 프로세서가 커널 모드인 동안 실행되는 것을 방지하기 위하여, 프로세서 코어와 캐시 간의 페이지 테이블 속성은 검사될 수 있다.

[166] 일 실시예에 따른 프로세서는, 주소 변환과 L1 캐시 액세스 간의 새로운 하드웨어 로직을 배제할 수 있다. 물리 주소를 검사하는 로직을 추가하는 것은, 필연적으로 게이트 및 와이어 지연을 증가시키고, 잠재적으로 다른 파이프라인 단계를 도입할 수 있다. 그 결과, 상당한 성능 저하가 발생될 수 있다. 예를 들어, Rocket Chip Generator의 PMP 구현은, 물리 주소를 검사하는 위치에 로직을 추가하지 않기 위하여 노력할 수 있다. 전술한 바와 같이, 일 실시예에 따른 프로세서는 MMU 또는 L1 캐시 대신에, 페이지 테이블 워커를 확장함으로써 로직을 추가하지 않을 수 있다.

[167] 결과적으로, 일 실시예에 따른 프로세서는, 메모리 액세스에 대하여 주소 변환과 함께 MMU에 의해 업데이트된 TLB 엔트리를 사용하여 검사될 수 있다. MMU는 주소 변환을 위한 프로세서 코어 및 캐시 사이에 놓여있어서, 모든 액세스 요청들은 캐시에서 메모리 접근 TLB 히트가 될지라도 일 실시예에 따른 프로세서의 정책을 위반하면 보호 위반(protection fault)를 유발할 수 있다. MMU 및 L1 캐시 사이에 새로운 하드웨어를 추가하지 않고도, 오로지 페이지 테이블 워커에 추가적인 하드웨어만을 추가함으로써 변형 없는 MMU가 활용될 수 있다.

[168] 일 실시예에 따른 프로세서는, 페이지 테이블 엔트리의 페이지 테이블 속성을 검사하기 위한 기존 하드웨어 로직의 이점을 활용함으로써, TLB 미스마다 페이지 테이블 엔트리(또는 페이지 테이블 엔트리의 페이지 테이블 속성)를 업데이트(예: 소독)함으로써 원하는 정책을 시행할 수 있다.

[169] 일 실시예에 따른 프로세서는, 운영 체제 커널이 변경되는 것이 요구될 수 있다. 예를 들어, 페이지 오류 핸들러에 대한 변경이 요구될 수 있다. 컴퓨팅 시스템들에서, 페이지 오류들은 요청 페이징(demand paging)의 과정에서, 정기적으로 발생할 수 있다. 추가 가상 페이지에 대한 어플리케이션의 요청들은 가상 페이지와 함께 먼저 제공되고, 물리적 페이지는 페이지가 터치될 때 프로비저닝(provision)될 수 있다.

[170] 여기에서, TLB가 인식하고 운영 체제 커널이 핸들링하는 페이지 오류가 포함될 수 있다. 페이지 폴트 후, 어플리케이션은 페이지 폴트를 유발한 명령어로부터 실행을 재개할 수 있다. 일 실시예에 따른 프로세서를 사용하기 위하여, 커널 코드 페이지에 대한 쓰기가 동일한 페이지 폴트를 유발하기 때문에, 페이지 폴트 핸들링 절차가 업데이트되어야 할 수 있다. 커널이 쓰기 가능한 가상 페이지를 커널 코드 페이지에 매핑하고 이에 쓰는 경우, 일 실시예에 따른 프로세서는 TLB 미스 시 페이지 테이블 속성을 업데이트하고 액세스가 거부되도록 유발할 수 있다. 커널이 이 오류를 핸들링하는 동안, 페이지가 쓰기 가능한지 확인한 후 오류가 처리된 것으로 간주하고 오류가 발생한 명령에서 실행을 재개할 수 있다. 그러나, 페이지 테이블 엔트리에 페이지가 쓰기 가능하다고 표시되어 있음에도, 일 실시예에 따른 프로세서가 페이지 테이블 엔트리를 다시 삭제하기 때문에 동일한 보호 오류가 다시 발생할 수 있다.

[171] 페이지 오류를 유발한 가상 주소가 커널 코드 페이지 내의 물리적 주소를 가리키는지를 커널이 확인하도록 페이지 오류 핸들러가 조정될 수 있다. 커널은 가상 주소에 대응하는 페이지 테이블 엔트리가 유효한 경우 일 실시예에 따른 프로세서의 커널 코드 페이지 보호에 의해 페이지 폴트가 발생한 것으로 판단할 수 있고, 액세스는 페이지 속성을 위반하지 않으며, 진입점은 커널 코드 페이지를 가리킬 수 있다. 커널은 보안 정책에 따라 오류를 일으킨 프로세스를 종료하거나 쓰기를 조용히 무시할 수 있다.

[172] 이하, 일 실시예에 따른 프로세서의 성능을 비교 실시예에 따른 프로세서와 비교함으로써 설명한다.

[173] 일 실시예에 따른 프로세서는, 예시적으로, Freedom U500 Dev Kit의 Rocket Chip Generator를 확장함으로써 구현될 수 있고, 일 실시예에 따른 프로세서가 성능, 에너지 소비, 칩 면적 및 임계 경로 지연(critical path latency)에 미치는 영향은 Xilinx VCU118 평가 키트를 사용하여 측정될 수 있다. 어플리케이션(예: Beebs 및 SPEC CPU 2006) 및 운영 체제 커널 벤치마크들을 이용한 실험들에서, 일 실시예에 따른 프로세서는 어플리케이션(<0.5%) 및 운영 체제(<3%) 성능에 낮은 영향을 미칠 수 있다. 칩 면적(0.13%) 및 에너지 소비(<2%) 측면에서 비용도 낮고, 임계 경로 지연을 증가시키지 않을 수 있다.

[174] Rocket Chip Generator가 확장됨으로써 일 실시예에 따른 프로세서(PRIVLOCK)가 구현될 수 있고 Freedom U500 V707 FPGA 개발 키트를 통해 FPGA에서 시스템이 평가될 수 있다. 특히 Xilinx VCU118 평가 키트용으로 개발 키트의 포트가 사용될 수 있다. 예시적으로, 커밋 ID는 943ab4ac2cefbbabdeda9447ec0f6231f6235f1e일 수 있다. Rocket Chip의 구성으로, 4개의 RockerTile들을 가지는 U500 플랫폼에 대한 기본값(default)가 사용될 수 있고, 각각의 RockerTiles은 하나의 Rocket 코어, 16KB의 L1 데이터 캐시, 및 16KB의 L1 명령어 캐시를 포함할 수 있다. 예를 들어, 커널 프로그램 및 사용자 프로그램을 컴파일 하기 위해 RISC-V 프로세서용 GNU 툴체인이 사용될 수 있고, 커밋 ID는 b4dae89f85bf882852c6186b1284df11065bfcd9일 수 있다. 프로토타입은 2GB의 외부 메모리로 실행되고 100MHz에서 작동하며 Linux 커널 버전 4.15.0을 실행할 수 있다. 프로토타입은 U500 플랫폼에서 수정되지 않은 Rocket Chip의 기본(default) 구성을 따라 100MHz에서 작동하도록 구성될 수 있다.

[175] 도 10은 일 실시예에 따른 프로세서 및 비교 실시예에 따른 프로세서의 성능을 LMBench를 이용하여 측정한 결과를 설명하기 위한 도면이다.

[176] 도 10에 나타난 바와 같이, 일 실시예에 따른 프로세서(PRIVLOCK)의 성능 오버헤드는 눈에 띄지 않을 수 있다. 보호 오류는 약 30% 정도 느려지지만 애플리케이션 벤치마크에서 볼 수 있듯이 간헐적 작동이기 때문에 시스템 성능에 크게 영향을 미치지 않을 수 있다. 모든 벤치마크에 대해 4가지 이전 작업들의 오버헤드는, 결과가 보고되지 않았기 때문에, 표시할 수 없었을 수 있다.

[177] 일 실시예에 따른 프로세서(PRIVLOCK)은, 전술한 바와 같이, 운영 체제 커널의 페이지 오류 핸들러를 계측할 수 있다. 따라서, 추가적인 체크가 메모리 관리 동작들 및 관련 운영 체제 서비스들의 성능에 영향을 미칠 가능성이 있다. 널리 사용되는 LMBench를 통해 일 실시예에 따른 프로세서(PRIVLOCK)이 운영 체제 성능에 미치는 잠재적 영향이 측정될 수 있다. 그림 3은 유사한 보안 보장을 제공하는 비교 실시예들에 따른 프로세서의 오버헤드와 함께 결과를 보여줄 수 있다. 예상대로, 일 실시예에 따른 프로세서(PRIVLOCK)은 운영 체제 성능도 또한 저하시키지 않을 수 있다. 참고로, 후술하겠으나, 보호 오류(protection fault)의 30% 오버헤드는 대부분의 어플리케이션들에서 눈에 띄지 않을 수 있다. 프로그램은 일반적으로 보호 오류 수를 줄이기 위해 최적화되어 있기 때문일 수 있다.

[178] 도 10에서, 비교 실시예들에 따른 프로세서에서 보고된 성능 효과가 나타날 수 있다. 일 실시예에 따른 프로세서(PRIVLOCK)의 오버헤드는 소프트웨어 전용 메커니즘(SecVisor 및 Nested Kernel)보다 상당히 낮고 하드웨어 지원 메커니즘(Kargos 및 RiskiM)과 유사할 수 있다. LMBench로 평가된 커널 코드 무결성을 보장하는 두 가지 소프트웨어 전용 메커니즘인 SecVisor 및 Nested Kernel은 각각 최대 10배 및 3배의 오버헤드를 가질 수 있다. 이것은 일 실시예에 따른 프로세서(PRIVLOCK)의 오버헤드보다 훨씬 클 수 있다. Kargos 및 RiskiM 커널 코드 무결성 위반을 감지할 수 있는 하드웨어 지원 메커니즘입니다. 둘 다(각각 약 3% 및 30%) 일 실시예에 따른 프로세서(PRIVLOCK)과 비교하여 유사하거나 더 나은 성능 오버헤드를 가질 수 있다.

[179] 소프트웨어 전용 메커니즘과 비교하여 일 실시예에 따른 프로세서(PRIVLOCK)의 잠재적인 이점을 강조하기 위해, 일 실시예에 따른 프로세서(PRIVLOCK)의 컨텍스트 전환 오버헤드는 에뮬레이트된 소프트웨어 전용 보호와 비교될 수 있다. 페이지 테이블 무결성의 확정적인 소프트웨어 전용 보호는 각 컨텍스트 스위치에서 새로 사용된 페이지 테이블들의 소독(sanitization)를 요구하고, 컨텍스트 스위치 대기 시간을 증가시킬 수 있다. 16개의 페이지 테이블들을 포함하는 모든 컨텍스트 스위치에서 16페이지(64KB)를 읽도록 커널은 계측되고 소프트웨어 전용 메커니즘이 에뮬레이트될 수 있다.

[180] 표 1은 LMBench로 측정된 컨텍스트 스위치 대기 시간의 결과를 나타낼 수 있다.

[181] [표 1]

[182]

Processes	Baseline	PRIVLOCK		Software-Only	
2	239ms	250ms	(1.05×)	5994ms	(23.9×)
16	315ms	325ms	(1.03×)	4466ms	(13.7×)
64	314ms	324ms	(1.03×)	1998ms	(6.16×)

[183] 각 컨텍스트 스위치에서 확인해야 하는 페이지 수는 항상 16보다 크므로, 실험에서 16페이지를 조사하는 것이 선택될 수 있다. 따라서 최소 예상 오버헤드가 제시된 수에 반영될 수 있다. 모든 컨텍스트 스위치마다, 새 페이지 테이블에서 유효한 모든 페이지 테이블 엔트리들을 검사해야 할 수 있고, 페이지 테이블이 데이터 페이지를 포함하여 64MB 이상의 메모리를 매핑하는 경우 페이지 테이블은 유효한 페이지 테이블 엔트리를 포함하는 적어도 16페이지들을 가져야 할 수 있다. 결과에서 나타난 바와 같이, 커널이 수행하는 추가 체크는 일 실시예에 따른 프로세서(PRIVLOCK)과 비교하여 컨텍스트 스위치들의 속도를 크게 저하시킬 수 있다.

[184] 도 11은 일 실시예에 따른 비교 실시예에 따른 프로세서를 가지는 시스템의 실행 시간으로 정규화된 일 실시예에 따른 프로세서를 가지는 시스템의 워크로드의 실행 시간을 도시한다.

[185] 많은 수의 페이지 폴트들을 유발하지 않는 어플리케이션들의 경우, 일 실시예에 따른 프로세서(PRIVLOCK)가 성능 저하를 유발하지 않을 것이 예상될 수 있다. 후술하겠으나, 추가 하드웨어는 소개할 페이지 테이블 워커에서 일부 경로들의 대기 시간을 증가시킬 수 있고, 주기 측면에서 작업의 대기 시간을 증가시키지 않을 수 있다. 첫 번째 어플리케이션 성능 연구에서, MiBench, WCET 벤치마크 및 DSPStone로부터 파생된 임베디드 시스템들을 평가하기 위하여 설계된 벤치마크 제품군(benchmark suite)인 Bristol/Embecosm Embedded Benchmark Suite(Beebs)[39]가 사용될 수 있다. 나머지 2개의 워크로드들은 메모리 안전 버그들을 가지는 것으로 확인되어 실행되지 않기 때문에, 제품군에서 80개의 워크로드들 중 78개가 실행될 수 있다. 도 11은 일 실시예에 따른 프로세서(PRIVLOCK)이 활성화된 경우 일 실시예에 따른 프로세서(PRIVLOCK)이 없는 시스템의 실행 시간으로 정규화된 각 워크로드의 실행 시간을 나타낼 수 있다. 각 워크로드는 10번 실행되고, 평균이 계산되고, 정규화된 실행 시간이 계산될 수 있다. 예상된 대로, 성능 오버헤드는 눈에 띄지 않을 수 있고, 0.5% 범위 내에서 유지되며, 기하평균에서는 0.3% 미만일 수 있다.

[186] 도 12는 일 실시예에 따른 SPEC 2006 벤치마크 제품군의 12개의 워크로드들의 정규화된 실행 시간을 도시한다.

[187] 일 실시예에 따른 프로세서(PRIVLOCK)의 보호 하에 있는 중요하지 않은 대규모 프로그램의 성능을 평가하기 위하여, SPEC 2006 벤치마크 제품군이 이용될 수 있다. 프로토타입의 제한된 컴퓨팅 성능과 파일 시스템 크기 때문에, 최신 버전인 SPEC 2017을 실행할 수 없을 수 있다. 같은 이유로, 보다 더 큰 ref 입력보다 작은 traininput이 이용될 수 있다. 도 12에서 나타난 바와 같이, 일 실시예에 따른 프로세서(PRIVLOCK)이 있는 시스템은 성능 오버헤드가 거의 0(예: -0.07%)인 사소하지 않은 프로그램을 실행할 수 있다.

[188] 표 2는 일 실시예에 따른 프로세서의 칩 영역(chip area)에 대한 영향(impact)를 설명한다.

[189] [표 2]

[190]

Module Name	Baseline	PRIVLOCK	
Top	22.90M	22.92M	(0.11%)
RocketTile	5.61M	5.62M	(0.12%)
Rocket	127.23K	131.25K	(3.16%)
CSRFile	30.18K	33.08K	(9.59%)
TLB	35.01K	35.17K	(0.46%)
PTW	14.37K	16.11K	(12.06%)
PRIVLOCKCHECKER	0	0.59K	

[191] 일 실시예에 따른 프로세서(PRIVLOCK)은 추가 하드웨어 리소스의 이점을 활용하기 때문에, 상당한 성능 저하 없이 추가 보안 보장을 제공합니다. 일 실시예에 따른 프로세서(PRIVLOCK)은, 이러한 추가 구성 요소가 없었으면, 페이지 테이블 컨텐츠를 확인하기 위해 더 많은 CPU 명령어들을 실행하고 더 높은 성능 오버헤드를 가졌을 수 있다. 추가 자원의 비용은 여러 가지 방법으로 평가될 수 있다. ASIC(Application Specific Integrated Circuit) 흐름과 FPGA 활용에 따른 칩 면적이 측정될 수 있다. 에너지 소비의 잠재적 비용은 FPGA에서 실행 중인 프로토타입과 구현된 디자인의 예상 전력 소비를 추정하는 Vivado Design Suite의 기능 모두에 의해 측정될 수 있다.

[192] 일 실시예에 따른 프로세서(PRIVLOCK)을 가지는 시스템이 ASIC으로 구현될 때, 칩 면적의 추정을 계산하기 위하여 Yosys 오픈 합성 제품군(Yosys)이 사용될 수 있다. 셀 라이브러리에서, FreePDK45[48]에 포함된 무료로 사용 가능한 라이브러리가 사용될 수 있다. 표 2는 관심 있는 여러 하드웨어 구성 요소들의 추정에 따라 상위 모듈과 전체 시스템의 면적 추정을 나타낼 수 있다. 전반적으로, 일 실시예에 따른 프로세서(PRIVLOCK)이 4개의 범위 레지스터들을 가질 때, 면적 비용은 0.13%에 불과할 수 있다. 4개의 범위 레지스터들을 가지는 일 실시예에 따른 프로세서(PRIVLOCK)은 성능 및 보안 평가에 사용될 수 있다. 표 2는 대부분의 면적 비용이 Rocket 코어 아래 CSRFile의 추가 레지스터들과 페이지 테이블 워커(PTW)의 커널 코드 영역 판단부(PRIVLOCKCHECKER)에서 발생하는 것을 나타낼 수 있다. 숫자에서 알 수 있듯이, 일 실시예에 따른 프로세서(PRIVLOCK)을 구현한 시스템은 주변 장치가 거의 없을 수 있다. 4개의 RocketTiles은 대부분의 칩 면적을 차지할 수 있다. 실제로, 시스템에 훨씬 더 많은 주변 장치가 있을 것으로 예상될 수 있으므로, 일 실시예에 따른 프로세서(PRIVLOCK)의 상대적 면적 비용을 훨씬 더 적게 만들 수 있다.

[193] 표 3은 일 실시예에 따른 프로세서의 임계 경로 지연(critical path latency)을 설명한다.

[194] [표 3]

[195]

Module Name	Baseline	PRIVLOCK
Rocket	2224.25	2277.45
CSRFile	1040.39	1027.95
TLB	860.18	873.5
PTW	1052.02	968.31
PRIVLOCKCHECKER	-	306.37

[196] 칩 면적 외에도, 일 실시예에 따른 프로세서(PRIVLOCK)이 임계 경로 지연을 크게 증가시키고 지연 제약을 완화시키는지 알아보기 위하여, 일부 관련 모듈들의 임계 경로 지연은 수집되고 비교될 수 있다. 표 3은 관련 모듈들의 임계 경로 지연을 나타낸다. 테스트된 모든 구성들에서, 일 실시예에 따른 프로세서(PRIVLOCK)은 베이스라인(baseline) 시스템이 충족하는 지연 제약 조건 2500ps를 위반하지 않을 수 있다. 관련 모듈의 임계 경로도 크게 증가하지 않을 수 있다.

[197] 표 4는 일 실시예에 따른 프로세서의 FPGA 리소스 활용(FPGA Resource Utilization)을 설명한다.

[198] [표 4]

[199]

Site Type	Baseline	PRIVLOCK	
Total LUTs	128.06K	128.76K	(0.54%)
Logic LUTs	122.79K	123.79K	(0.57%)
LURAMs	4.60K	4.60K	(0.0%)
SRLs	0.67K	0.67K	(0.0%)
FFs	75.10%	75.78%	(0.90%)
RAMB36	0.03%	0.03%	(0%)
RAM18	0.19%	0.19%	(0%)
URAM	0.00%	0.00%	(0%)
DSP48 Blocks	0.06%	0.06%	(0%)

[200] 잠재적인 면적 비용에 대한 또 다른 추정으로서, 추정된 칩 면적 외에 FPGA 활용에 대한 일 실시예에 따른 프로세서(PRIVLOCK)의 영향이 제시될 수 있다. 표 4는 자원 활용 측면에서도 FPGA에서 구현될 때, 일 실시예에 따른 프로세서(PRIVLOCK)의 면적 비용이 작은(<3%) 것을 나타낼 수 있다.

[201] 도 13은 일 실시예에 따른 프로세서의 정규화된 추가적인 에너지 소모를 설명하기 위한 도면이다.

[202] 표 5는 일 실시예에 따른 프로세서의 추가적인 에너지 소모를 평가하기 위하여 이용된 각 세트의 구성(composition)을 설명한다.

[203] 표 6은 일 실시예에 따른 프로세서의 FPGA 구현되는 경우 추정된 전력 소모를 설명한다.

[204] [표 5]

[205]

SET 1	SET 2	SET 3	SET 4	SET 5
aha-compress	ctl-string	fasta	lcdnum	ndes
aha-mont64	ctl-vector	fdct	levenshtein	nettle-aes
bs	cubic	fibcall	ludcmp	nettle-arcfour
bubblesort	dijkstra	fir	matmult-float	nettle-cast128
cnt	dtoa	frac	matmult-int	nettle-des
compress	duff	huffbench	mergesort	nettle-md5
cover	edn	insertsort	miniz	nettle-sha256
crc	expint	janne_complex	minver	newlib-exp
ctl-stack	fac	jfdctint	nbody	newlib-log
SET 6	SET 7	SET 8	SET 9	
newlib-mod	recursion	sglib-listsort	strstr	
newlib-sqrt	rijndael	sglib-queue	tarai	
ns	select	sglib-rbtree	template	
nsichneu	sglib-arraybinsearch	slre	trio-snprintf	
picojpeg	sglib-arrayheapsort	sqrt	trio-sscanf	
prime	sglib-arrayquicksort	st	ud	
qrduino	sglib-dllist	statemate	whetstone	
qsort	sglib-hashtable	stb_perlin	wikisort	
qurt	sglib-listinsertsort	stringsearch1		

[206] [표 6]

[207]

Module Name	Baseline	PRIVLOCK
Top	1.834W	1.848W(0.8%)
PTW	25mW	39mW(56%)

[208] 추가 하드웨어 구성 요소들은, 시스템이 실행되는 동안 추가 에너지를 소비할 수 있다. 두 가지 방법을 통해 에너지 소비의 예상 비용은 추정될 수 있다. 일 실시예에 따른 프로세서(PRIVLOCK)을 사용하여 FPGA 구현 시스템에서 Beebs 벤치마크 제품군의 하위 집합을 실행하는 동안 에너지 소비가 측정될 수 있다.

[209] 폴링(polling) 기반 전력 측정의 제한된 세분성으로 인해, 워크로드 세트의 에너지 소비가 측정될 수 있다. 표 5는 각 세트에 포함된 워크로드를 설명할 수 있다. 결과(예: 기하학적 평균에서 1.87%)는 일 실시예에 따른 프로세서(PRIVLOCK)에 대해 새로 추가된 하드웨어 구성 요소들이 상당한 에너지 오버헤드를 유발하지 않음이 시사될 수 있다. FPGA에서 측정을 보완하기 위한 또 다른 측정으로, 시스템이 FPGA에서 구현될 때, 예상되는 전력 소비를 획득하기 위해 Xilinx

Vivado[57]의 빌트인 전력 추정 기능이 사용될 수 있다. 표 6은 에너지 측정의 결론과 유사하게 추가 전력 소비가 작을 것(<1%)으로 예상되는 것을 보여줄 수 있다.

[210] 커널 코드 무결성을 변조하는 가능한 공격을 보여주는 세 가지 합성 공격들이 구현될 수 있고, 일 실시예에 따른 프로세서(PRIVLOCK)은 모든 합성 공격들을 성공적으로 방지할 수 있다.

[211] 첫 번째 공격은, 쓰기가 불법으로 인식되는지 확인하기 위하여, 커널 코드 페이지 중 하나에 대한 쓰기 액세스를 만들 수 있다. 이 공격은, 기존의 메커니즘이 전통적인 페이지 테이블 기반 보호를 사용하여 어려움 없이 방지할 수 있지만, 커널 코드 페이지가 기본적으로 쓰기 가능하도록 구성되어 있기 때문에 테스트된 시스템에서 공격이 성공할 수 있다. 이러한 페이지 테이블 속성들의 부족과 관계없이, 일 실시예에 따른 프로세서(PRIVLOCK)이 인에이블링될 때 공격이 중지되고 커널에 보호 오류가 발생할 수 있다.

[212] 두 번째는 공격자가 악성 코드를 커널 데이터 페이지에 저장하고 실행을 시도하는 시나리오에 대하여 작성될 수 있다. 공격자는 코드가 포함된 페이지를 준비할 뿐만 아니라 대응하는 페이지 테이블 엔트리들을 조작하여 프로세스가 특권 모드(privileged mode)에서 동작하는 동안 페이지를 실행 가능하게 만들어야 할 수 있다. 이 절차는 성공적으로 구현될 수 있고, 보호되지 않은 시스템은 공격을 감지하지 못할 수 있다. 첫 번째 작업과 달리, 커널은 이전 작업에서 제안한 것처럼 페이지 테이블들의 직접적인 조작을 포함하는 절차를 방지할 수 없을 수 있다. 일 실시예에 따른 프로세서(PRIVLOCK)은 이러한 행위를 검출하도록 설계되지 않았지만, 프로세서가 물리적 커널 코드 페이지 외부에 위치된 악성 코드로 점프할 때 공격을 방지하기 때문에, 일 실시예에 따른 프로세서(PRIVLOCK)은 페이지 테이블 조작 단계에서 이 공격을 검출하거나 방지할 수 없을 수 있다.

[213] 세 번째 공격은 커널 코드 페이지들을 수정하는 것을 목표로 하지만, 커널 코드 페이지에 대해 새로 생성된 가상 페이지를 사용한다는 점에서 첫 번째 공격보다 고급일 수 있다. 첫 번째 공격과 달리, 운영 체제 커널은 악성 페이지 테이블 수정을 방지하거나 완화하기 위한 일부 메커니즘들[5, 13]로 강화되지 않는 한, 이 공격을 검출할 수 없을 수 있다. 예상된 대로, 구현된 공격은 일 실시예에 따른 프로세서(PRIVLOCK)이 인에이블링되지 않을 때 커널 코드 페이지를 성공적으로 손상시키지만, 일 실시예에 따른 프로세서(PRIVLOCK)은 커널 코드 페이지가 덮어쓰일 때 공격을 방지할 수 있다. 추가 가상 페이지가 있더라도, 쓰기는 물리적 커널 코드 페이지로 전달될 수 있다. 공격은 새 매핑을 쓰기 가능한 페이지로 생성하지만, 일 실시예에 따른 프로세서(PRIVLOCK)은 엔트리가 TLB에 로드되고 프로세서가 이 페이지를 쓰기 방지된 것으로 인식할 때 이 속성을 클리어링(clear) 수 있다.

[214] 이상에서 설명된 실시예들은 하드웨어 구성요소, 소프트웨어 구성요소, 및/또는 하드웨어 구성요소 및 소프트웨어 구성요소의 조합으로 구현될 수 있다. 예를

들어, 실시예들에서 설명된 장치, 방법 및 구성요소는, 예를 들어, 프로세서, 콘트롤러, ALU(arithmetic logic unit), 디지털 신호 프로세서(digital signal processor), 마이크로컴퓨터, FPGA(field programmable gate array), PLU(programmable logic unit), 마이크로프로세서, 또는 명령(instruction)을 실행하고 응답할 수 있는 다른 어떠한 장치와 같이, 범용 컴퓨터 또는 특수 목적 컴퓨터를 이용하여 구현될 수 있다. 처리 장치는 운영 체제(OS) 및 상기 운영 체제 상에서 수행되는 소프트웨어 애플리케이션을 수행할 수 있다. 또한, 처리 장치는 소프트웨어의 실행에 응답하여, 데이터를 접근, 저장, 조작, 처리 및 생성할 수도 있다. 이해의 편의를 위하여, 처리 장치는 하나가 사용되는 것으로 설명된 경우도 있지만, 해당 기술분야에서 통상의 지식을 가진 자는, 처리 장치가 복수 개의 처리 요소(processing element) 및/또는 복수 유형의 처리 요소를 포함할 수 있음을 알 수 있다. 예를 들어, 처리 장치는 복수 개의 프로세서 또는 하나의 프로세서 및 하나의 컨트롤러를 포함할 수 있다. 또한, 병렬 프로세서(parallel processor)와 같은, 다른 처리 구성(processing configuration)도 가능하다.

[215] 소프트웨어는 컴퓨터 프로그램(computer program), 코드(code), 명령(instruction), 또는 이들 중 하나 이상의 조합을 포함할 수 있으며, 원하는 대로 동작하도록 처리 장치를 구성하거나 독립적으로 또는 결합적으로(collectively) 처리 장치를 명령할 수 있다. 소프트웨어 및/또는 데이터는, 처리 장치에 의하여 해석되거나 처리 장치에 명령 또는 데이터를 제공하기 위하여, 어떤 유형의 기계, 구성요소(component), 물리적 장치, 가상 장치(virtual equipment), 컴퓨터 저장 매체 또는 장치, 또는 전송되는 신호 파(signal wave)에 영구적으로, 또는 일시적으로 구체화(embody)될 수 있다. 소프트웨어는 네트워크로 연결된 컴퓨터 시스템 상에 분산되어서, 분산된 방법으로 저장되거나 실행될 수도 있다. 소프트웨어 및 데이터는 컴퓨터 판독 가능 기록 매체에 저장될 수 있다.

[216] 실시예에 따른 방법은 다양한 컴퓨터 수단을 통하여 수행될 수 있는 프로그램 명령 형태로 구현되어 컴퓨터 판독 가능 매체에 기록될 수 있다. 컴퓨터 판독 가능 매체는 프로그램 명령, 데이터 파일, 데이터 구조 등을 단독으로 또는 조합하여 포함할 수 있으며 매체에 기록되는 프로그램 명령은 실시예를 위하여 특별히 설계되고 구성된 것들이거나 컴퓨터 소프트웨어 당업자에게 공지되어 사용 가능한 것일 수도 있다. 컴퓨터 판독 가능 기록 매체의 예에는 하드 디스크, 플로피 디스크 및 자기 테이프와 같은 자기 매체(magnetic media), CD-ROM, DVD와 같은 광기록 매체(optical media), 플롭티컬 디스크(floptical disk)와 같은 자기-광 매체(magneto-optical media), 및 롬(ROM), 램(RAM), 플래시 메모리 등과 같은 프로그램 명령을 저장하고 수행하도록 특별히 구성된 하드웨어 장치가 포함된다. 프로그램 명령의 예에는 컴파일러에 의해 만들어지는 것과 같은 기계어 코드뿐만 아니라 인터프리터 등을 사용해서 컴퓨터에 의해서 실행될 수 있는 고급 언어 코드를 포함한다.

[217] 위에서 설명한 하드웨어 장치는 실시예의 동작을 수행하기 위해 하나 또는 복수의 소프트웨어 모듈로서 작동하도록 구성될 수 있으며, 그 역도 마찬가지이다.

[218] 이상과 같이 실시예들이 비록 한정된 도면에 의해 설명되었으나, 해당 기술분야에서 통상의 지식을 가진 자라면 이를 기초로 다양한 기술적 수정 및 변형을 적용할 수 있다. 예를 들어, 설명된 기술들이 설명된 방법과 다른 순서로 수행되거나, 및/또는 설명된 시스템, 구조, 장치, 회로 등의 구성요소들이 설명된 방법과 다른 형태로 결합 또는 조합되거나, 다른 구성요소 또는 균등물에 의하여 대치되거나 치환되더라도 적절한 결과가 달성될 수 있다.

[219] 그러므로, 다른 구현들, 다른 실시예들 및 특허청구범위와 균등한 것들도 후술하는 특허청구범위의 범위에 속한다.

청구범위

[청구항 1] 프로세서에 의하여 수행되는 커널 코드 영역에 대한 동작을 제어하기 위한 방법에 있어서,
프로세서가 머신 모드(machine mode)인 동안, 커널 코드(kernel code)가 저장된 커널 코드 영역의 물리 주소(physical address)에 기초하여 범위 레지스터(range register)를 설정(configure)하는 단계;
상기 프로세서가 커널 모드(kernel mode)인 동안 페이지(page)의 가상 주소에 기초한 액세스 요청을 수신하는 경우에 응답하여, 상기 페이지의 상기 가상 주소에 대한 페이지 테이블 엔트리(page table entry)를 이용하여 상기 페이지의 물리 주소를 획득하는 단계;
상기 설정된 범위 레지스터를 상기 획득된 물리 주소와 비교함으로써, 상기 획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는지 여부를 판단하는 단계; 및
상기 획득된 물리 주소에 대응하는 영역이 커널 코드 영역(kernel code region)에 포함되는지 여부에 기초하여, 상기 물리 주소에 대응하는 영역에 대한 쓰기 동작 또는 실행 동작 중 적어도 하나의 동작에 대한 인에이블(enable) 또는 디스에이블(disable) 여부를 결정하는 단계
를 포함하는 방법.

[청구항 2] 제1항에 있어서,
상기 쓰기 동작 또는 실행 동작 중 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정하는 단계는,
상기 획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는 경우에 응답하여, 상기 물리 주소에 대응하는 영역에 대한 쓰기 동작(write operation)을 디스에이블하는 단계를 포함하는,
방법.

[청구항 3] 제1항에 있어서,
상기 쓰기 동작 또는 실행 동작 중 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정하는 단계는,
상기 획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는 경우에 응답하여, 상기 프로세서가 커널 모드인 동안 상기 물리 주소에 대응하는 영역에 대한 실행 동작(execute operation)을 인에이블하는 단계를 포함하는,
방법.

[청구항 4] 제1항에 있어서,
상기 쓰기 동작 또는 실행 동작 중 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정하는 단계는,

상기 획득된 물리 주소에 대응하는 영역이 커널 코드 영역과 다른 영역에 포함되는 경우에 응답하여, 상기 프로세서가 커널 모드인 동안 상기 물리 주소에 대응하는 영역에 대한 실행 동작(execute operation)을 디스에이블(disable)하는 단계를 포함하는,
방법.

[청구항 5] 제1항에 있어서,
상기 쓰기 동작 또는 실행 동작 중 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정하는 단계는,
상기 페이지 테이블 엔트리의 페이지 테이블 속성(page table attribute)과 독립적으로, 상기 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정하는 단계를 포함하는,
방법.

[청구항 6] 제1항에 있어서,
상기 범위 레지스터를 설정하는 단계는,
상기 프로세서가 머신 모드인 동안, 커널 코드를 가지는 페이지를 미리 결정된 크기를 가지는 청크(chunk)의 경계(boundary)에 따라 정렬(align)하는 단계를 포함하는,
방법.

[청구항 7] 제1항에 있어서,
상기 범위 레지스터를 설정하는 단계는,
상기 커널 코드를 적어도 하나의 커널 코드 청크(kernel code chunk)로 저장하는 단계; 및
각 커널 코드 청크(kernel code chunk)에 대하여, 해당 커널 코드 청크에 대응하는 범위 레지스터의 적어도 일부를 설정하는 단계를 포함하는,
방법.

[청구항 8] 제1항에 있어서,
상기 범위 레지스터를 설정하는 단계는,
각 커널 코드 청크의 시작 물리 주소에 기초하여, 해당 커널 코드 청크에 대응하는 범위 레지스터의 베이스 비트(base bit)를 설정하는 단계; 및
해당 커널 코드 청크의 크기에 기초하여, 해당 커널 코드 청크에 대응하는 범위 레지스터의 마스크 비트(mask bit)를 설정하는 단계를 포함하는,
방법.

[청구항 9] 제1항에 있어서,
상기 범위 레지스터를 설정하는 단계는,
상기 범위 레지스터의 베이스 비트 및 마스크 비트를 설정한 경우에 응답하여, 상기 범위 레지스터의 밸리드 비트(valid bit) 및 락 비트(lock bit)를 설정하는 단계를 포함하는,
방법.

[청구항 10] 제1항에 있어서,
상기 페이지의 물리 주소를 획득하는 단계는,
변환 색인 버퍼(translation lookaside buffer; TLB)에 상기 액세스 요청의 상기 가상 주소에 대한 상기 페이지 테이블 엔트리가 캐시(cache)되어 있는지 여부를 확인하는 단계를 포함하고,
상기 방법은,
상기 변환 색인 버퍼에 상기 페이지 테이블 엔트리가 캐시되어 있는 경우에 응답하여, 상기 설정된 범위 레지스터와 상기 획득된 물리 주소의 비교를 생략하는 단계
를 더 포함하는 방법.

[청구항 11] 제1항에 있어서,
변환 색인 버퍼에 상기 페이지 테이블 엔트리가 캐시되어 있는 경우에 응답하여, 상기 페이지 테이블 엔트리의 페이지 테이블 속성에 기초하여 상기 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정하는 단계
를 더 포함하는 방법.

[청구항 12] 제1항에 있어서,
상기 페이지의 물리 주소를 획득하는 단계는,
상기 변환 색인 버퍼에 상기 페이지 테이블 엔트리가 캐시되어 있지 않은 경우에 응답하여, 페이지 테이블로부터 상기 가상 주소에 대한 상기 페이지 테이블 엔트리를 검색하는 단계; 및
상기 검색된 페이지 테이블 엔트리에 기초하여, 상기 가상 주소에 매핑된 상기 페이지의 상기 물리 주소를 획득하는 단계를 포함하는,
방법.

[청구항 13] 제1항에 있어서,
상기 결정된 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부에 기초하여, 상기 페이지 테이블 엔트리의 페이지 테이블 속성을 업데이트하는 단계; 및
상기 업데이트된 페이지 테이블 속성을 상기 페이지 테이블 엔트리와 함께 변환 색인 버퍼에 캐시하는 단계
를 더 포함하는 방법.

[청구항 14] 제1항에 있어서,
상기 획득된 물리 주소에 대응하는 영역 커널 코드 영역에 포함되는지 여부를 판단하는 단계는,
상기 범위 레지스터의 마스크 비트에 기초하여, 상기 물리 주소 중 부분 주소를 결정하는 단계; 및

상기 결정된 부분 주소가 상기 범위 레지스터의 베이스 비트의 적어도 일부에 매칭되는 경우 상기 획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는 것을 판단하는 단계를 포함하는,
방법.

[청구항 15] 하드웨어와 결합되어 제1항 내지 제14항 중 어느 하나의 항의 방법을 실행시키기 위하여 컴퓨터 판독 가능한 기록매체에 저장된 컴퓨터 프로그램.

[청구항 16] 커널 코드 영역에 대한 동작을 제어하기 위한 전자 장치에 있어서,
프로세서가 머신 모드(machine mode)인 동안, 커널 코드(kernel code)가 저장된 커널 코드 영역의 물리 주소(physical address)에 기초하여 범위 레지스터(range register)를 설정(configure)하고, 상기 프로세서가 커널 모드(kernel mode)인 동안 페이지(page)의 가상 주소에 기초한 액세스 요청을 수신하는 경우에 응답하여, 상기 페이지의 상기 가상 주소에 대한 페이지 테이블 엔트리(page table entry)를 이용하여 상기 페이지의 물리 주소를 획득하며, 상기 설정된 범위 레지스터를 상기 획득된 물리 주소와 비교함으로써, 상기 획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는지 여부를 판단하고, 상기 획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는지 여부에 기초하여, 상기 물리 주소에 대응하는 영역에 대한 쓰기 동작 또는 실행 동작 중 적어도 하나의 동작에 대한 인에이블(enable) 또는 디스에이블(disable) 여부를 결정하는 프로세서
를 포함하는 전자 장치.

[청구항 17] 제16항에 있어서,
상기 프로세서는,
상기 쓰기 동작 또는 실행 동작 중 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정하고,
상기 획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는 경우에 응답하여, 상기 물리 주소에 대응하는 영역에 대한 쓰기 동작(write operation)을 디스에이블하는,
전자 장치.

[청구항 18] 제16항에 있어서,
상기 프로세서는,
상기 쓰기 동작 또는 실행 동작 중 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정하고,
상기 획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는 경우에 응답하여, 상기 프로세서가 커널 모드인 동안 상기 물리 주소에 대응하는 영역에 대한 실행 동작(execute operation)을 인에이블하는,
전자 장치.

[청구항 19] 제16항에 있어서,

상기 프로세서는,
상기 쓰기 동작 또는 실행 동작 중 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정하고,
상기 획득된 물리 주소에 대응하는 영역이 커널 코드 영역과 다른 영역에 포함되는 경우에 응답하여, 상기 프로세서가 커널 모드인 동안 상기 물리 주소에 대응하는 영역에 대한 실행 동작(excute operation)을 디스에이블(disable)하는,
전자 장치.

[청구항 20] 제16항에 있어서,
상기 프로세서는,
상기 쓰기 동작 또는 실행 동작 중 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정하고,
상기 페이지 테이블 엔트리의 페이지 테이블 속성(page table attribute)과 독립적으로, 상기 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정하는,
전자 장치.

[청구항 21] 제16항에 있어서,
상기 프로세서는,
상기 프로세서가 머신 모드인 동안, 커널 코드를 가지는 페이지를 미리 결정된 크기를 가지는 청크(chunk)의 경계(boundary)에 따라 정렬(align)하는,
전자 장치.

[청구항 22] 제16항에 있어서,
상기 프로세서는,
상기 커널 코드를 적어도 하나의 커널 코드 청크(kernel code chunk)로 저장하고,
각 커널 코드 청크(kernel code chunk)에 대하여, 해당 커널 코드 청크에 대응하는 범위 레지스터의 적어도 일부를 설정하는,
전자 장치.

[청구항 23] 제16항에 있어서,
상기 프로세서는,
각 커널 코드 청크의 시작 물리 주소에 기초하여, 해당 커널 코드 청크에 대응하는 범위 레지스터의 베이스 비트(base bit)를 설정하고,
해당 커널 코드 청크의 크기에 기초하여, 해당 커널 코드 청크에 대응하는 범위 레지스터의 마스크 비트(mask bit)를 설정하는,
전자 장치.

[청구항 24] 제16항에 있어서,
상기 프로세서는,

상기 범위 레지스터의 베이스 비트 및 마스크 비트를 설정한 경우에 응답하여, 상기 범위 레지스터의 밸리드 비트(valid bit) 및 락 비트(lock bit)를 설정하는,
전자 장치.

[청구항 25] 제16항에 있어서,
상기 프로세서는,
변환 색인 버퍼(translation lookaside buffer; TLB)에 상기 액세스 요청의 상기 가상 주소에 대한 상기 페이지 테이블 엔트리가 캐시(cache)되어 있는지 여부를 확인하고,
상기 변환 색인 버퍼에 상기 페이지 테이블 엔트리가 캐시되어 있는 경우에 응답하여, 상기 설정된 범위 레지스터와 상기 획득된 물리 주소의 비교를 생략하는,
전자 장치.

[청구항 26] 제16항에 있어서,
상기 프로세서는,
변환 색인 버퍼에 상기 페이지 테이블 엔트리가 캐시되어 있는 경우에 응답하여, 상기 페이지 테이블 엔트리의 페이지 테이블 속성에 기초하여 상기 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정하는,
전자 장치.

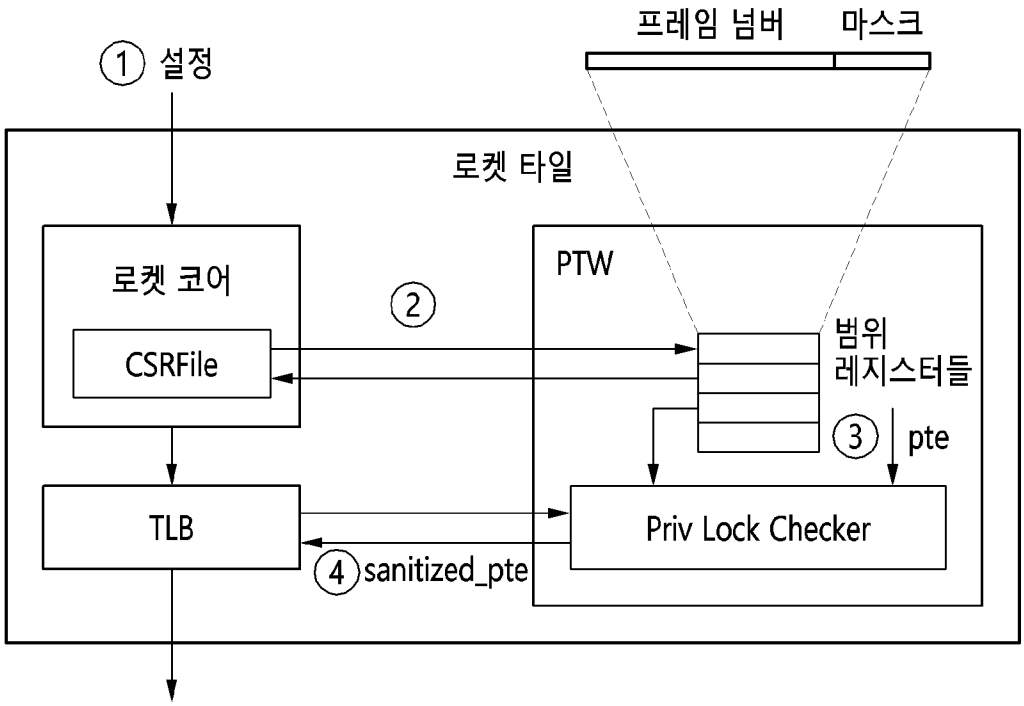
[청구항 27] 제16항에 있어서,
상기 프로세서는,
상기 변환 색인 버퍼에 상기 페이지 테이블 엔트리가 캐시되어 있지 않은 경우에 응답하여, 페이지 테이블로부터 상기 가상 주소에 대한 상기 페이지 테이블 엔트리를 검색하고,
상기 검색된 페이지 테이블 엔트리에 기초하여, 상기 가상 주소에 매핑된 상기 페이지의 상기 물리 주소를 획득하는,
전자 장치.

[청구항 28] 제16항에 있어서,
상기 프로세서는,
상기 결정된 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부에 기초하여, 상기 페이지 테이블 엔트리의 페이지 테이블 속성을 업데이트하고,
상기 업데이트된 페이지 테이블 속성을 상기 페이지 테이블 엔트리와 함께 변환 색인 버퍼에 캐시하는,
전자 장치.

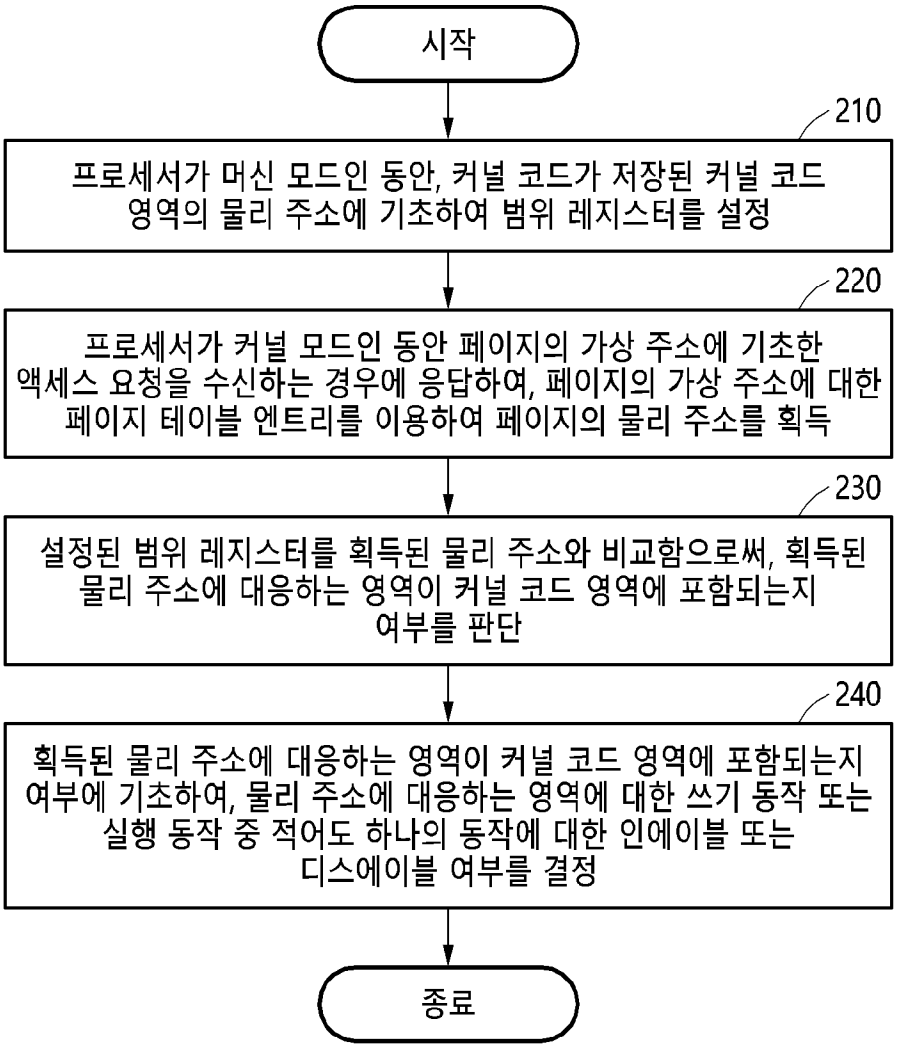
[청구항 29] 제16항에 있어서,
상기 프로세서는,

상기 범위 레지스터의 마스크 비트에 기초하여, 상기 물리 주소 중 부분 주소를 결정하고,
상기 결정된 부분 주소가 상기 범위 레지스터의 베이스 비트의 적어도 일부에 매칭되는 경우 상기 획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는 것을 판단하는,
전자 장치.

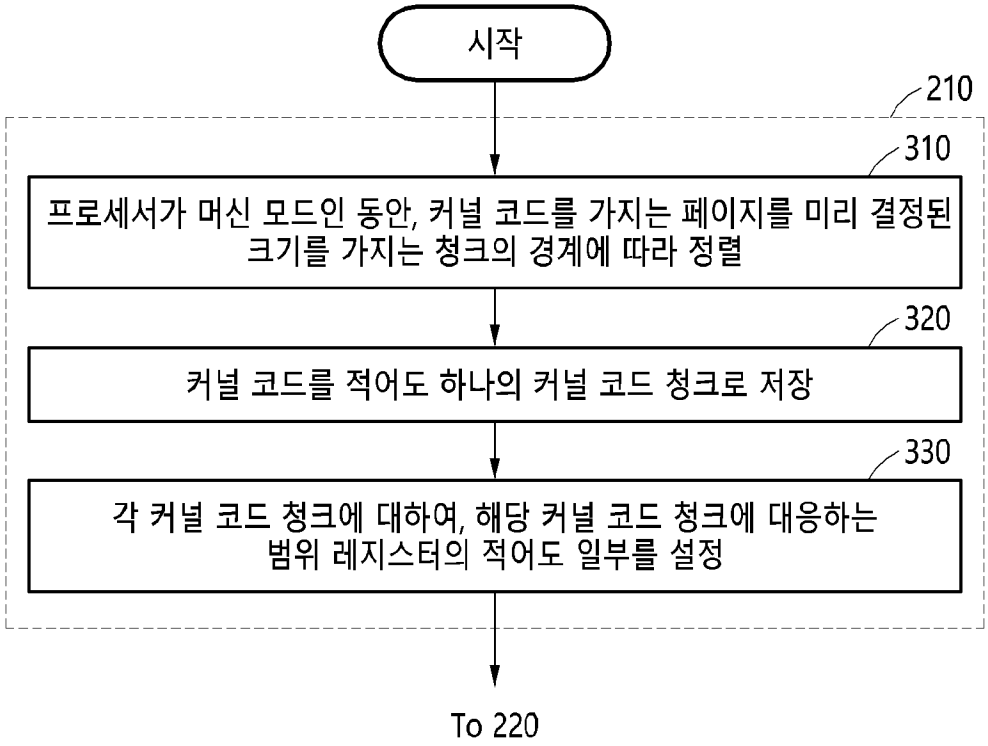
[도1]


프레임 넘버
마스크
① 설정
로켓 타일
로켓 코어
CSRFile
②
PTW
범위
레지스터들
③ pte
TLB
Priv Lock Checker
④ sanitized_pte

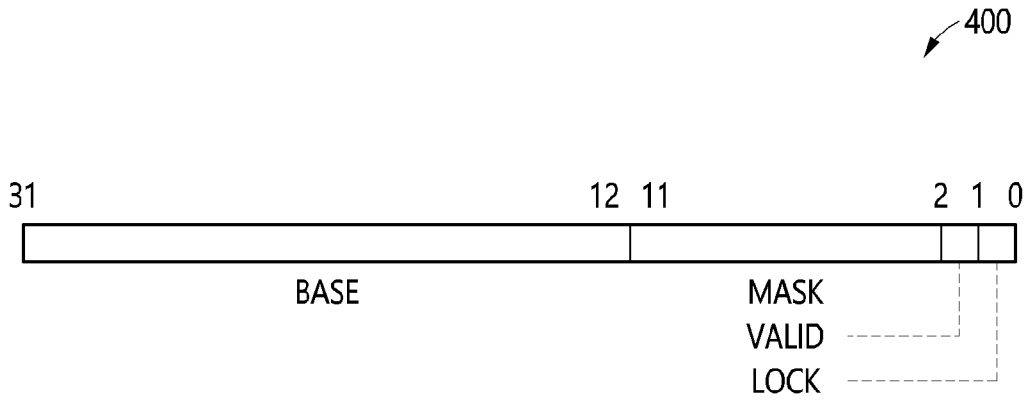
[도2]


시작
210
프로세서가 머신 모드인 동안, 커널 코드가 저장된 커널 코드 영역의 물리 주소에 기초하여 범위 레지스터를 설정
220
프로세서가 커널 모드인 동안 페이지의 가상 주소에 기초한 액세스 요청을 수신하는 경우에 응답하여, 페이지의 가상 주소에 대한 페이지 테이블 엔트리를 이용하여 페이지의 물리 주소를 획득
230
설정된 범위 레지스터를 획득된 물리 주소와 비교함으로써, 획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는지 여부를 판단
240
획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는지 여부에 기초하여, 물리 주소에 대응하는 영역에 대한 쓰기 동작 또는 실행 동작 중 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정
종료

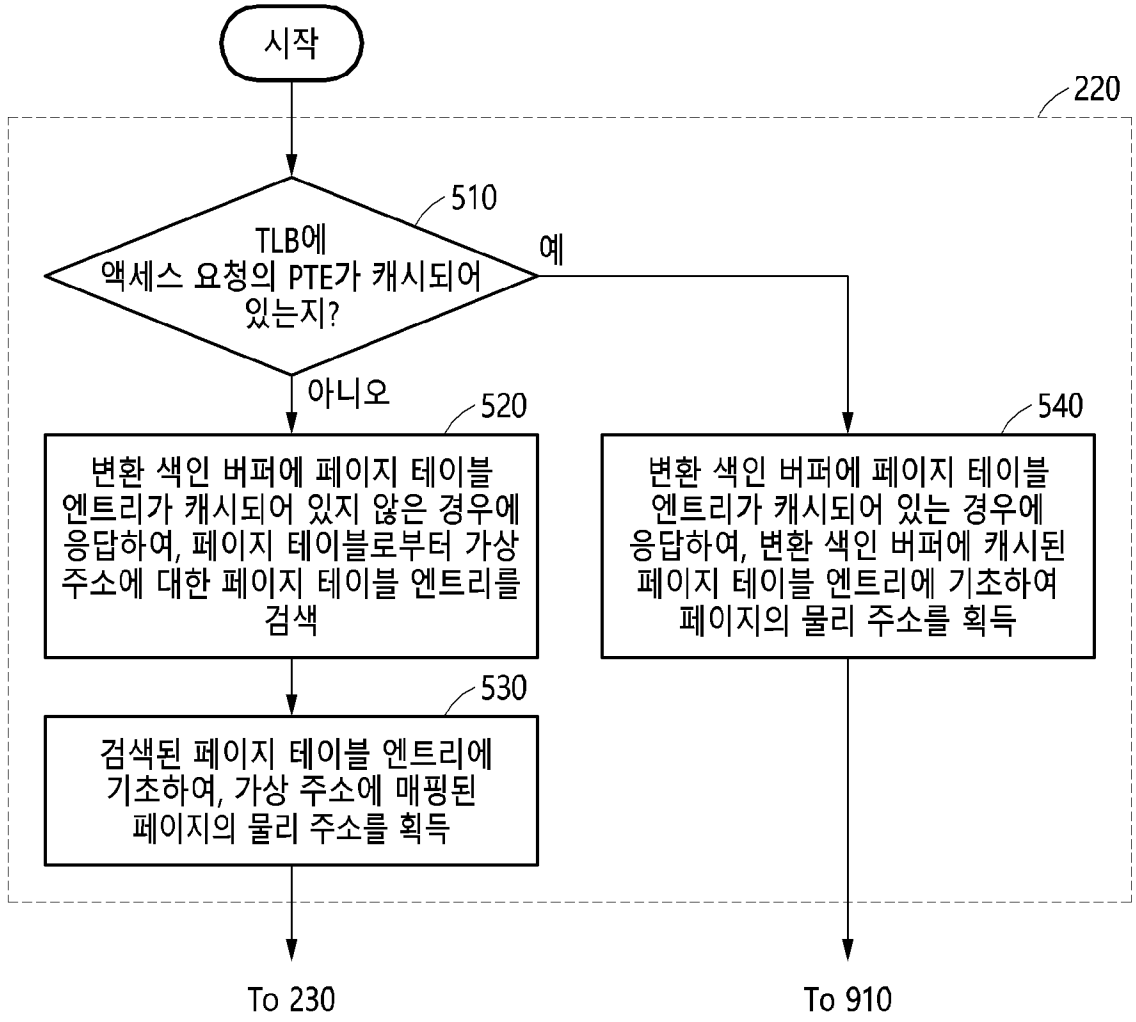
[도3]


시작
210
310
프로세서가 머신 모드인 동안, 커널 코드를 가지는 페이지를 미리 결정된 크기를 가지는 청크의 경계에 따라 정렬
320
커널 코드를 적어도 하나의 커널 코드 청크로 저장
330
각 커널 코드 청크에 대하여, 해당 커널 코드 청크에 대응하는 범위 레지스터의 적어도 일부를 설정
To 220

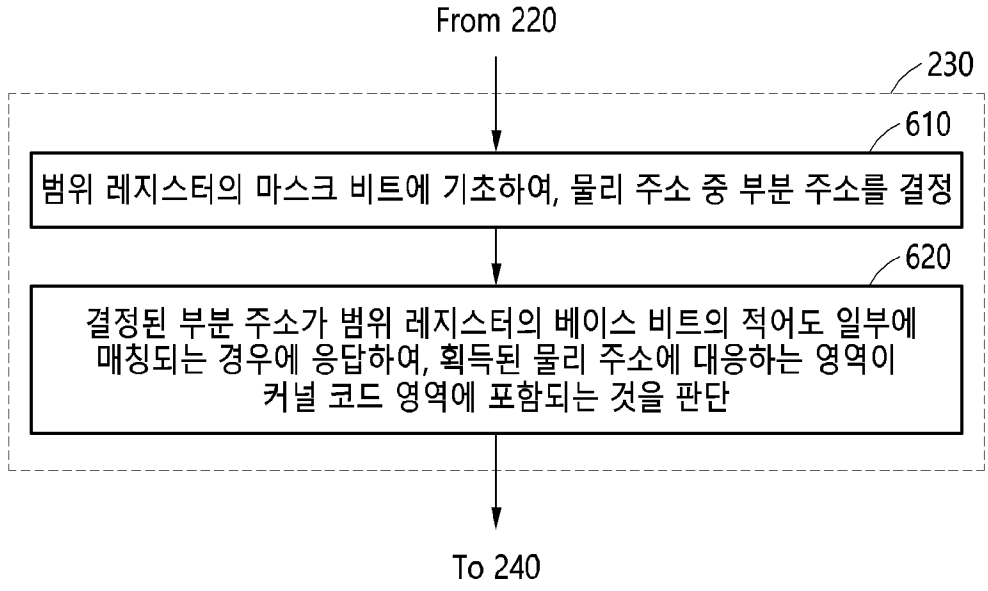
[도4]


400
31
12
11
2
1
0
BASE
MASK
VALID
LOCK

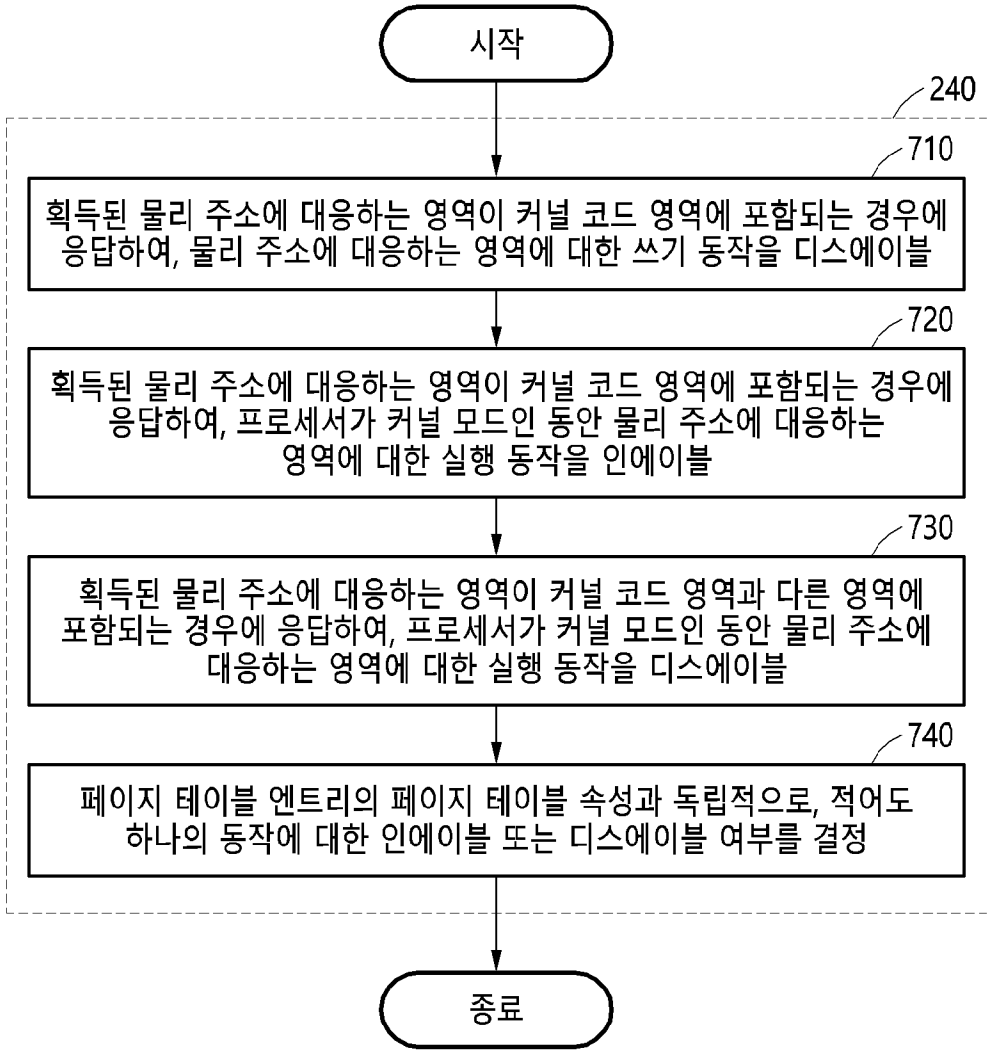
[도5]


시작
220
510
TLB에
액세스 요청의 PTE가 캐시되어
있는지?
예
아니오
520
변환 색인 버퍼에 페이지 테이블
엔트리가 캐시되어 있지 않은 경우에
응답하여, 페이지 테이블로부터 가상
주소에 대한 페이지 테이블 엔트리를
검색
530
검색된 페이지 테이블 엔트리에
기초하여, 가상 주소에 매핑된
페이지의 물리 주소를 획득
540
변환 색인 버퍼에 페이지 테이블
엔트리가 캐시되어 있는 경우에
응답하여, 변환 색인 버퍼에 캐시된
페이지 테이블 엔트리에 기초하여
페이지의 물리 주소를 획득
To 230
To 910

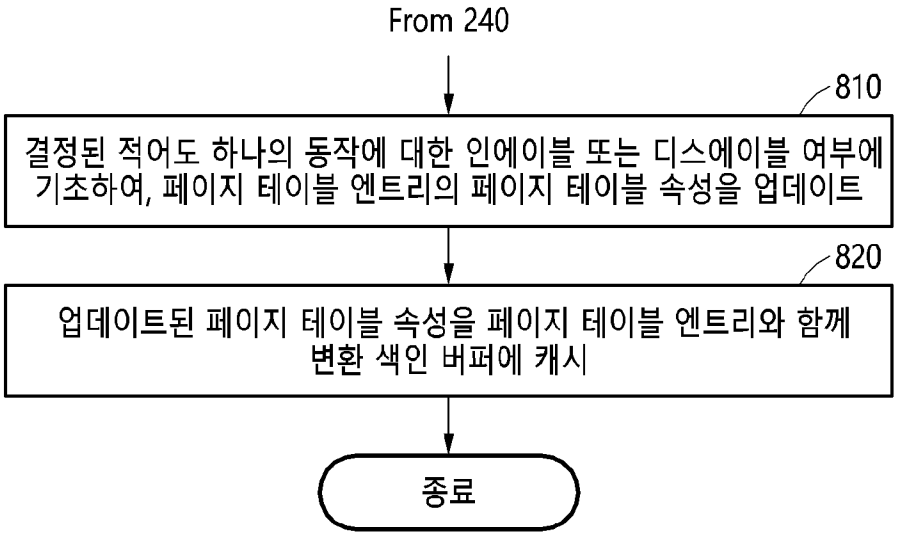
[도6]


From 220
230
610
범위 레지스터의 마스크 비트에 기초하여, 물리 주소 중 부분 주소를 결정
620
결정된 부분 주소가 범위 레지스터의 베이스 비트의 적어도 일부에
매칭되는 경우에 응답하여, 획득된 물리 주소에 대응하는 영역이
커널 코드 영역에 포함되는 것을 판단
To 240

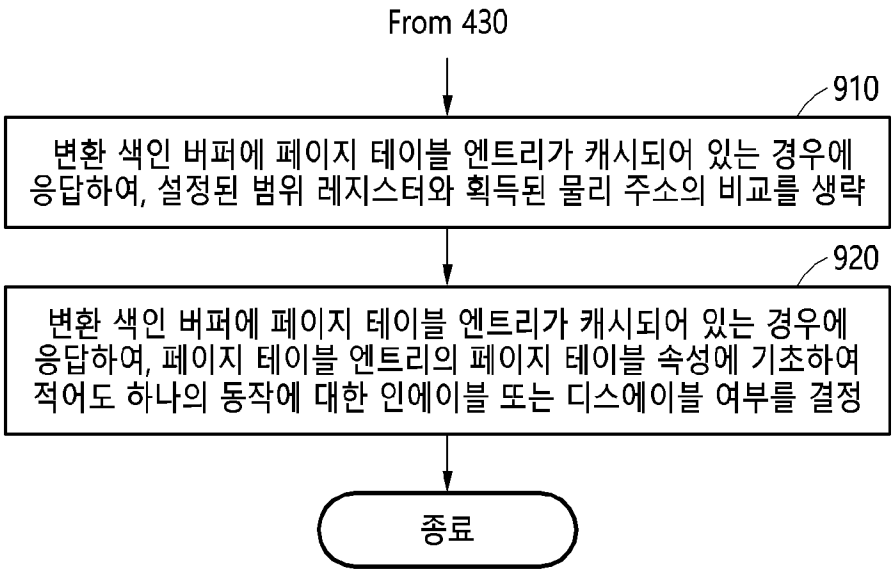
[도7]


시작
240
710
획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는 경우에 응답하여, 물리 주소에 대응하는 영역에 대한 쓰기 동작을 디스에이블
720
획득된 물리 주소에 대응하는 영역이 커널 코드 영역에 포함되는 경우에 응답하여, 프로세서가 커널 모드인 동안 물리 주소에 대응하는 영역에 대한 실행 동작을 인에이블
730
획득된 물리 주소에 대응하는 영역이 커널 코드 영역과 다른 영역에 포함되는 경우에 응답하여, 프로세서가 커널 모드인 동안 물리 주소에 대응하는 영역에 대한 실행 동작을 디스에이블
740
페이지 테이블 엔트리의 페이지 테이블 속성과 독립적으로, 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정
종료

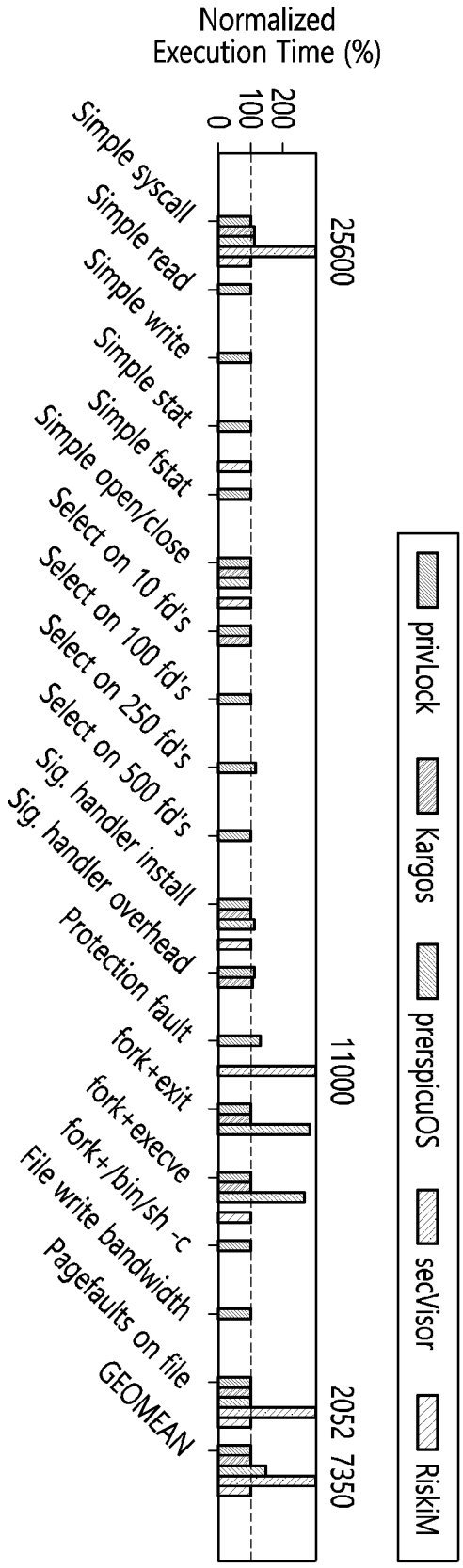
[도8]


From 240
810
결정된 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부에 기초하여, 페이지 테이블 엔트리의 페이지 테이블 속성을 업데이트
820
업데이트된 페이지 테이블 속성을 페이지 테이블 엔트리와 함께 변환 색인 버퍼에 캐시
종료

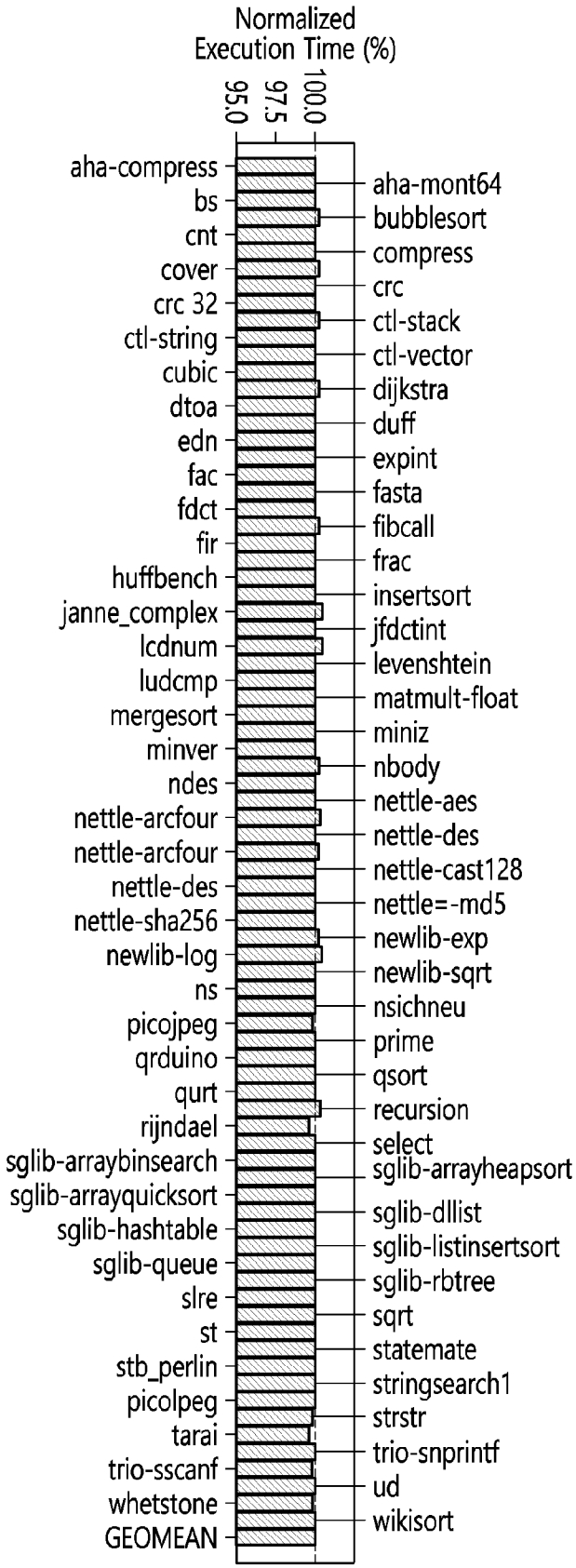
[도9]


From 430
910
변환 색인 버퍼에 페이지 테이블 엔트리가 캐시되어 있는 경우에 응답하여, 설정된 범위 레지스터와 획득된 물리 주소의 비교를 생략
920
변환 색인 버퍼에 페이지 테이블 엔트리가 캐시되어 있는 경우에 응답하여, 페이지 테이블 엔트리의 페이지 테이블 속성에 기초하여 적어도 하나의 동작에 대한 인에이블 또는 디스에이블 여부를 결정
종료

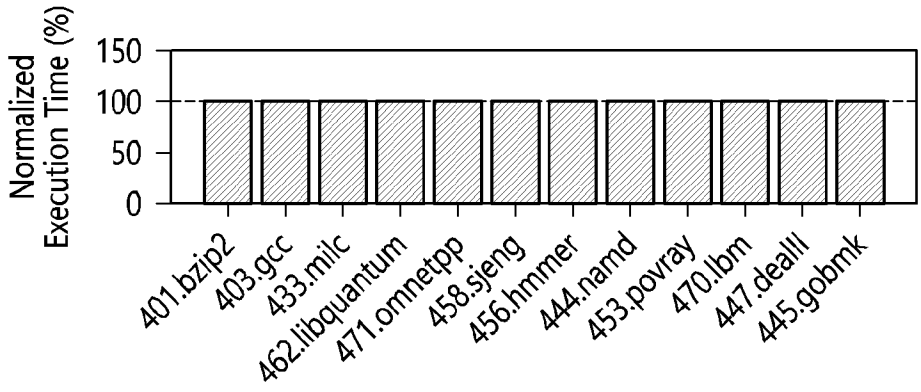
[도10]


Normalized
Execution Time (%)
0
100
200
Simple syscall
Simple read
Simple write
Simple stat
Simple fstat
Simple open/close
Select on 10 fd's
Select on 100 fd's
Select on 250 fd's
Select on 500 fd's
Sig. handler install
Sig. handler overhead
Protection fault
fork+exit
fork+execve
fork+/bin/sh -c
File write bandwidth
Pagefaults on file
GEOMEAN
25600
11000
2052
7350
privLock
Kargos
prerspicuOS
secVisor
RiskiM

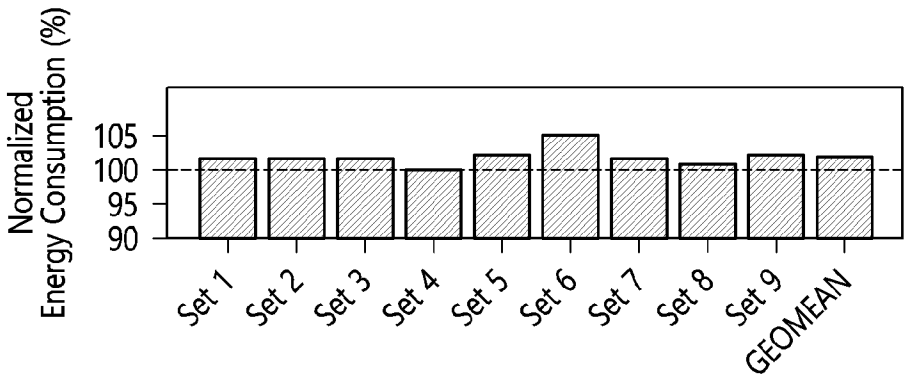
[도11]


Normalized
Execution Time (%)
95.0
97.5
100.0
aha-compress
bs
cnt
cover
crc 32
ctl-string
cubic
dtoa
edn
fac
fdct
fir
huffbench
janne_complex
lcdnum
ludcmp
mergesort
minver
ndes
nettle-arcfour
nettle-arcfour
nettle-des
nettle-sha256
newlib-log
ns
picojpeg
qrduino
qurt
rijndael
sglib-arraybinsearch
sglib-arrayquicksort
sglib-hashtable
sglib-queue
slre
st
stb_perlin
picolpeg
tarai
trio-sscanf
whetstone
GEOMEAN
aha-mont64
bubblesort
compress
crc
ctl-stack
ctl-vector
dijkstra
duff
expint
fasta
fibcall
frac
insertsort
jfdctint
levenshtein
matmult-float
miniz
nbody
nettle-aes
nettle-des
nettle-cast128
nettle=-md5
newlib-exp
newlib-sqrt
nsichneu
prime
qsort
recursion
select
sglib-arrayheapsort
sglib-dllist
sglib-listinsertsort
sglib-rbtree
sqrt
statemate
stringsearch1
strstr
trio-snprintf
ud
wikisort

[도12]


Normalized
Execution Time (%)
150
100
50
0
401.bzip2
403.gcc
433.milc
462.libquantum
471.omnetpp
458.sjeng
456.hmmer
444.namd
453.povray
470.lbm
447.dealII
445.gobmk

[도13]


Normalized
Energy Consumption (%)
105
100
95
90
Set 1
Set 2
Set 3
Set 4
Set 5
Set 6
Set 7
Set 8
Set 9
GEOMEAN

INTERNATIONAL SEARCH REPORT

International application No.
PCT/KR2023/007599

A. CLASSIFICATION OF SUBJECT MATTER

G06F 21/55(2013.01)i; **G06F 21/57**(2013.01)i; **G06F 21/53**(2013.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F 21/55(2013.01); G06F 12/10(2006.01); G06F 21/50(2013.01); G06F 9/455(2006.01)

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Korean utility models and applications for utility models: IPC as above
Japanese utility models and applications for utility models: IPC as above

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

eKOMPASS (KIPO internal) & keywords: RISC-V, 커널 코드(kernel code), 범위 레지스터(range register), 페이지 테이블 엔트리(page table entry), 물리 주소(physical address)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	KR 10-2020-0116311 A (SEOUL NATIONAL UNIVERSITY R&DB FOUNDATION) 12 October 2020 (2020-10-12) See paragraphs [0034]-[0040]; and claim 8.	1-29
A	KR 10-2017-0060815 A (SAMSUNG ELECTRONICS CO., LTD. et al.) 02 June 2017 (2017-06-02) See paragraphs [0041]-[0072]; claims 1-20; and figures 1-13.	1-29
A	US 2013-0283004 A1 (VMWARE, INC.) 24 October 2013 (2013-10-24) See paragraphs [0027]-[0089]; claims 1-20; and figures 1-11.	1-29
A	MA, Jun et al. Construction of RISC-V Lightweight Trusted Execution Environment Based on Hardware Extension. 2021 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS). pp. 237-242, 29 July 2021. [Retrieved on 17 August 2023]. Retrieved from <https://ieeexplore.ieee.org/abstract/document/9524261>. See pages 237-241; and figures 1-5.	1-29

☑ Further documents are listed in the continuation of Box C. ☑ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"D" document cited by the applicant in the international application
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
31 August 2023	**01 September 2023**

Name and mailing address of the ISA/KR	Authorized officer
Korean Intellectual Property Office **Government Complex-Daejeon Building 4, 189 Cheongsa-ro, Seo-gu, Daejeon 35208**	
Facsimile No. **+82-42-481-8578**	Telephone No.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	MOON, Hyungon et al. Architectural Supports to Protect OS Kernels from Code-Injection Attacks and Their Applications. ACM Transactions on Design Automation of Electronic Systems. Vol. 23, No. 1, pp. 1-25, 31 August 2017. [Retrieved on 17 August 2023]. Retrieved from <https://dl.acm.org/doi/abs/10.1145/3110223>. See pages 1-22; and figures 1-7.	1-29

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/KR2023/007599

Patent document cited in search report	Publication date (day/month/year)	Patent family member(s)	Publication date (day/month/year)
KR 10-2020-0116311 A	12 October 2020	KR 10-2183649 B1	26 November 2020
KR 10-2017-0060815 A	02 June 2017	KR 10-2494167 B1	01 February 2023
US 2013-0283004 A1	24 October 2013	US 2009-0300263 A1	03 December 2009
		US 2009-0300264 A1	03 December 2009
		US 2009-0300611 A1	03 December 2009
		US 2009-0300612 A1	03 December 2009
		US 2009-0300645 A1	03 December 2009
		US 2012-0047348 A1	23 February 2012
		US 8074045 B2	06 December 2011
		US 8086822 B2	27 December 2011
		US 8127107 B2	28 February 2012
		US 8245227 B2	14 August 2012
		US 8464022 B2	11 June 2013
		US 8868880 B2	21 October 2014
		US 9009727 B2	14 April 2015

국 제 조 사 보 고 서

국제출원번호
PCT/KR2023/007599

A. 발명이 속하는 기술분류(국제특허분류(IPC))

G06F 21/55(2013.01)i; **G06F 21/57**(2013.01)i; **G06F 21/53**(2013.01)i

B. 조사된 분야

조사된 최소문헌(국제특허분류를 기재)

G06F 21/55(2013.01); G06F 12/10(2006.01); G06F 21/50(2013.01); G06F 9/455(2006.01)

조사된 기술분야에 속하는 최소문헌 이외의 문헌

한국등록실용신안공보 및 한국공개실용신안공보: 조사된 최소문헌란에 기재된 IPC
일본등록실용신안공보 및 일본공개실용신안공보: 조사된 최소문헌란에 기재된 IPC

국제조사에 이용된 전산 데이터베이스(데이터베이스의 명칭 및 검색어(해당하는 경우))

eKOMPASS(특허청 내부 검색시스템) & 키워드: RISC-V, 커널 코드(kernel code), 범위 레지스터(range register), 페이지 테이블 엔트리(page table entry), 물리 주소(physical address)

C. 관련 문헌

카테고리*	인용문헌명 및 관련 구절(해당하는 경우)의 기재	관련 청구항
A	KR 10-2020-0116311 A (서울대학교산학협력단) 2020.10.12 단락 [0034]-[0040]; 및 청구항 8	1-29
A	KR 10-2017-0060815 A (삼성전자주식회사 등) 2017.06.02 단락 [0041]-[0072]; 청구항 1-20; 및 도면 1-13	1-29
A	US 2013-0283004 A1 (VMWARE, INC.) 2013.10.24 단락 [0027]-[0089]; 청구항 1-20; 및 도면 1-11	1-29
A	JUN MA 등, Construction of RISC-V Lightweight Trusted Execution Environment Based on Hardware Extension, 2021 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS), pp. 237-242, 2021.07.29 [검색일: 2023-08-17], 출처 <https://ieeexplore.ieee.org/abstract/document/9524261> 페이지 237-241; 및 도면 1-5	1-29

☑ 추가 문헌이 C(계속)에 기재되어 있습니다.

☑ 대응특허에 관한 별지를 참조하십시오.

* 인용된 문헌의 특별 카테고리:

"A" 특별히 관련이 없는 것으로 보이는 일반적인 기술수준을 정의한 문헌
"D" 본 국제출원에서 출원인이 인용한 문헌
"E" 국제출원일보다 빠른 출원일 또는 우선일을 가지나 국제출원일 이후에 공개된 선출원 또는 특허 문헌
"L" 우선권 주장에 의문을 제기하는 문헌 또는 다른 인용문헌의 공개일 또는 다른 특별한 이유(이유를 명시)를 밝히기 위하여 인용된 문헌
"O" 구두 개시, 사용, 전시 또는 기타 수단을 언급하고 있는 문헌
"P" 우선일 이후에 공개되었으나 국제출원일 이전에 공개된 문헌
"T" 국제출원일 또는 우선일 후에 공개된 문헌으로, 출원과 상충하지 않으며 발명의 기초가 되는 원리나 이론을 이해하기 위해 인용된 문헌
"X" 특별한 관련이 있는 문헌. 해당 문헌 하나만으로 청구된 발명의 신규성 또는 진보성이 없는 것으로 본다.
"Y" 특별한 관련이 있는 문헌. 해당 문헌이 하나 이상의 다른 문헌과 조합하는 경우로 그 조합이 당업자에게 자명한 경우 청구된 발명은 진보성이 없는 것으로 본다.
"&" 동일한 대응특허문헌에 속하는 문헌

국제조사의 실제 완료일	국제조사보고서 발송일
2023년08월31일(31.08.2023)	**2023년09월01일(01.09.2023)**
ISA/KR의 명칭 및 우편주소 **대한민국 특허청** **(35208) 대전광역시 서구 청사로 189, 4동 (둔산동, 정부대전청사)** 팩스 번호 **+82-42-481-8578**	심사관 **변성철** 전화번호 **+82-42-481-8262**

서식 PCT/ISA/210 (두 번째 용지) (2022년 7월)

국 제 조 사 보 고 서

국제출원번호
PCT/KR2023/007599

C. 관련 문헌

카테고리*	인용문헌명 및 관련 구절(해당하는 경우)의 기재	관련 청구항
A	HYUNGON MOON 등, Architectural Supports to Protect OS Kernels from Code-Injection Attacks and Their Applications, ACM Transactions on Design Automation of Electronic Systems, Vol. 23, No. 1, pp 1-25, 2017.08.31 [검색일: 2023-08-17], 출처 <https://dl.acm.org/doi/abs/10.1145/3110223> 페이지 1-22; 및 도면 1-7	1-29

국 제 조 사 보 고 서
대응특허에 관한 정보

국제출원번호
PCT/KR2023/007599

국제조사보고서에서 인용된 특허문헌	공개일	대응특허문헌	공개일
KR 10-2020-0116311 A	2020/10/12	KR 10-2183649 B1	2020/11/26
KR 10-2017-0060815 A	2017/06/02	KR 10-2494167 B1	2023/02/01
US 2013-0283004 A1	2013/10/24	US 2009-0300263 A1	2009/12/03
		US 2009-0300264 A1	2009/12/03
		US 2009-0300611 A1	2009/12/03
		US 2009-0300612 A1	2009/12/03
		US 2009-0300645 A1	2009/12/03
		US 2012-0047348 A1	2012/02/23
		US 8074045 B2	2011/12/06
		US 8086822 B2	2011/12/27
		US 8127107 B2	2012/02/28
		US 8245227 B2	2012/08/14
		US 8464022 B2	2013/06/11
		US 8868880 B2	2014/10/21
		US 9009727 B2	2015/04/14