DASICS功能手册 (version 2.1.0)

一、DASICS基本特征描述

DASICS(Dynamic in-Address-Space Isolation by Code Segments)是一种安全处理器设计方案，通过对不同代码片段访问的内存地址空间进行隔离并设置各自的访存权限，从而实现对非预期的越界访存和跳转的防护。这类越界访存可能来自于包括第三方恶意代码, 软件bug, 利用猜测执行的（如Spectre）漏洞在内的各种情况。DASICS的特点是通过硬件检查的方式对每条访存和跳转指令进行检查，在出现权限错误的时候触发用户态中断以处理访存或跳转错误：

1）硬件会根据当前PC所处的指令地址所在的代码片段范围判断当前的访存权限，即DASICS的权限是以代码片段作为一个主体（subject）单位进行划分的；

2）代码片段，是一段地址连续的代码，范围可以由权限更高的代码片段动态的在特殊寄存器中进行设置；

3）当前代码片段可访存地址是通过一组DASICS寄存器划分的多个地址空间片段，可以由权限更高的代码动态地修改。

4）在代码片段内部的跳转，以及通过DASICS寄存器组已经设定好的可访问地址范围内的数据访问，代码执行不受限制且保持高效率。

5）代码片段的指令如果访问超出设定的范围，则发生用户态（或同级）中断，由权限更高的代码片段进行处理。

DASICS的原名是IMPULP（In-process Memory Protection by User Lever Partitions），因此下面的部分命名依然存在PULP字样。

- 代码片段：地址空间连续的可执行代码的集合，通过起始、结束地址限定。代码片段是权限判定的主体。
- 主函数/可信区：具有最高权限的代码片段，可以修改当前系统状态（内核态、用户态、管理态）所允许的所有地址空间和寄存器，可以跳转到所有（可执行）代码入口，不会受到DASICS访问控制的影响。主函数负责动态建立其他代码片段的权限，并处理用户态（或同级）中断。
- 库函数/非可信区：地址跳转和数据访问受限的代码片段。库函数通常由权限更高的代码片段调用，并且只能返回到调用代码，其他合法跳转必须在库函数内部。库函数可访问的数据，由当前的边界寄存器组进行控制。

二、寄存器介绍

1、配置寄存器(config)

- DasicsSMainCfg(0xBC0)：控制S态主函数区的CSR寄存器，M态可配置
  - 寄存器位宽与其他CSR寄存器相同，均为XLEN，但目前仅有第0～2位bit被用到，其余位暂且保留。
  - 第0位为清零位(SCFG_CLS)，该位置一时，将会在下一个时钟周期时对除DasicsSMainCfg以外的DASICS寄存器进行清零，并将SCFG_CLS位清零。
  - 第1位为使能位(SCFG_ENA)，该位置一时，启动对S态的分区保护机制，即位于DasicsSMainLo与DasicsSMainHi之间的代码段将会被视为S态主函数区域；置零则关闭S态分区保护机制，即认为所有在S态执行的代码都是可信的，但U态的分区保护机制仍可以正常运行。
  - 第2位为全局使能位(SCFG_GLB)，该位置零时，关闭U态和S态的DASICS分区保护机制。
- DasicsUMainCfg(0x5C0)：控制U态主函数区的CSR寄存器，S态主函数区、M态可对其配置
  - 寄存器位宽与其他CSR寄存器相同，均为XLEN，但目前仅有第0、1位bit被用到，其余位保持reserved状态。
  - 第0位为清零位(UCFG_CLS)，该位置一时，将会在下一个时钟周期时对除DasicsSMainCfg、DasicsSMainBoundHi、DasicsSMainBoundLo、DasicsUMainCfg以外的DASICS寄存器进行清零，并将UCFG_CLS位清零。
  - 第1位为使能位(UCFG_ENA)，该位用来使能U态的DASICS保护机制，即位于DasicsUMainLo与DasicsUMainHi之间的代码段被视为U态的主函数区域。此外，操作系统可以在spawn的时候进行设置，从而创造可信进程。
- DasicsLibCfg0(0x881)、DasicsLibCfg1(0x882)：控制库函数/非可信区的CSR寄存器，U态主函数区、S态主函数区和M态可对其配置
  - DasicsLibCfg的设计模仿了PMP的config寄存器：每个DasicsLibCfg包含8个4-bit的config，每个config对应一组DasicsLibBound寄存器。当CSR寄存器位宽XLEN=64时，每个DasicsLibCfg偶数位编号的config不会使用（编号从1开始）。
  - 对于单个config：
    - 第3位为有效位(LIBCFG_V)，表示该config对应的LibBound是否生效；
    - 第2位为执行位(LIBCFG_X)，在目前的设计中，该位置1时，LibBound对应的区域成为自由跳转区，从而无须设置额外的自由跳转区寄存器；
    - 第1位为可读位(LIBCFG_R)，表示该config对应的LibBound内的数据能否被库函数读取；
    - 第0位为可写位(LIBCFG_W)，表示该config对应的LibBound内的数据能否被库函数写入。

2、限界寄存器(bound)

- DasicsSMainBoundHi(0xBC1), DasicsSMainBoundLo(0xBC2): S态主函数代码区上下界寄存器，M态、使用sbi调用的S态主函数可配置
  - 目前的实现中，由于BBL与ucas-os独立编译，无法访问到ucas-os的信息，因此S态主函数区边界先由M态的BBL在minit.c中进行静态配置（配置为全空间），再在进入S态主函数区范围之后，由主函数区代码进行sbi调用（sbi_modify_smain_bound）来更新寄存器。此外，在该调用的实现中，会先判断发起调用的代码是否为S态主函数区，如果不是的话会报错终止程序。

    - S态的主函数在链接的时候，会被集中放置在.smaintext段中。因此，S态主函数通过sbi调用来设置DasicsSMainBoundHi与DasicsSMainBoundLo分别为该段的上下界。
- DasicsUMainBoundHi(0x5C1)、DasicsUMainBoundLo(0x5C2): U态主函数区上下界寄存器，M态、S态主函数可配置
    - U态的可信函数在链接的时候，会被集中放置在.umaintext段中。因此，S态主函数可以根据该段的上下界来配置DasicsUMainBoundHi与DasicsUMainBoundLo两个寄存器。
- DasicsLibBound(0x883~0x8A2): 一共16组32个库函数边界寄存器，M态、S态主函数、U态主函数可配置
    - 在0x883~0x8A2这32个寄存器中，由低地址到高地址按照(Hi, Lo)来排列库函数边界寄存器，即第i组的库函数边界寄存器，其上界编号为0x883 + i * 2，下界编号为0x884 + i * 2，并且与DasicsLibCfg0（i<8）或者DasicsLibCfg1（i>=8）中的config相对应。

3、控制流&主函数调用相关寄存器

- DasicsReturnPC(0x8A4)：主函数调用库函数返回地址寄存器
    - 在主函数调用库函数时，硬件会自动保存调用结束时的返回地址(PC+4)。如果程序从库函数跳转回主函数区的目标地址不为DasicsReturnPC所记录的返回地址，也不为主函数调用入口地址，则引发DASICS跳转例外。
    - 该寄存器设置为可读写，是因为在主函数调用的入口中，需要暂时保存寄存器的值，并在调用结束时恢复。
- DasicsFreeZoneReturnPC(0x8A5): 自由跳转区返回地址寄存器
    - 当控制流由库函数非自由跳转区跳转进入库函数自由跳转区时，硬件会自动保存库函数非自由跳转区的调用返回地址(Pc+4)。如果程序从库函数自由跳转区返回库函数非自由跳转区时的目标地址不为DasicsFreeZoneReturnPC所记录的返回地址，则会引发DASICS跳转例外。
    - 该寄存器设置为可读写，理由同DasicsReturnPC
- DasicsMaincallEntry(0x8A3)：主函数调用入口寄存器
    - 该寄存器可以在程序上下文切换的时候进行复用，即在发生程序上下文切换前的U态时，该寄存器为U态主函数调用的入口地址；在发生程序上下文切换后的S态时，该寄存器为S态主函数调用的入口地址。
    - 在S态主函数和U态主函数中，程序可以自己定义主函数调用入口函数，并设置DasicsMaincallEntry的值为该函数的起始地址。
    - 主函数调用模仿了系统调用，为库函数提供了一个可信区代码的调用入口，同时可以在入口函数中对传入的调用参数进行检查。
    - 在ucas-os的项目实现中，默认的主函数调用入口的工作流程为：
        - 由于入口函数之后要跳转到其他主函数区函数，而这些主函数区的函数又会调用其他的库函数，从而会覆盖掉先前保存的返回地址，因此需要先保存DasicsReturnPC和DasicsFreeZonePC的值。
        - 之后入口函数根据传入的调用号跳转并执行对应的主函数区函数。

    - 执行完成后，恢复先前保存的DasicsReturnPC和DasicsFreeZonePC的值，并使用pulpret指令返回。

4、用户态中断相关寄存器、指令、例外号

- 除CYCLE、TIME、INSTRET以及浮点相关的用户态CSR寄存器以外，实现了RV-Draft(https://github.com/riscv/riscv-isa-manual/releases/tag/draft-20210612-98f3349)中的N扩展，以及sedeleg、sideleg两个代理寄存器。
- 除sedeleg和sideleg由M态、S态主函数配置以外，其他诸如utval的寄存器由M态、S态主函数、U态主函数进行配置。
- 实现了uret指令，用于用户态中断的例外返回。
- 新增了6种DASICS例外，分别如下表所示。可以通过修改BBL初始化m/s代理寄存器的逻辑来将例外代理到用户态处理。

DASICS 例外名	例外号	备注
DasicsUFetchFault	0x18	U态跳转例外
DasicsSFetchFault	0x19	S态跳转例外
DasicsULoadFault	0x1a	U态无读权限
DasicsSLoadFault	0x1b	S态无读权限
DasicsUStoreFault	0x1c	U态无写权限
DasicsSStoreFault	0x1d	S态无写权限

三、PULP指令介绍

- 目前的DASICS项目只增加了IMPULP项目的一条PULPRET指令，其余诸如PULPMRW的PULP指令都通过复用CSR模块读写逻辑、复用LibBound作为自由跳转区边界等方法予以消除。
- PULPRET: 由可信区返回非可信区时，不修改调用库函数返回地址寄存器，用于支持主函数调用的功能。
    - PULPRET实现时复用了jalr的数据通路，本质上是一条特殊的jalr指令
    - PULPRET在小尾端机器上的指令码为.word 0x0000f00b；如果使用118上经过修改的riscv-7pulp-elf编译器，可以将上述指令码替换为pulpret x0,0,x1

四、主要功能介绍

1、 内核加载进程

- **含义**：进程启动时需要由内核设置好主函数代码区上下界，这样才能使得后面的代码执行具有权限，保证后续的主函数区和库函数区权限设置正常。
- **流程**：ucas-os的链接器脚本对.text和.data进行了分区。内核根据主函数代码段(.smaintext, .umaintext)的上下界分别设置S态和U态的主函数代码区上下界寄存器，把加载函数映射到S态主函数区，由加载函数负责加载elf文件（合理放置文件中的各个段，代码里库函数区指定为.slibtext .ulibtext，自由跳转区指定为.sfreezonetext .ufreezonetext）

2、 主函数调用库函数

- **含义**：需要保证跳转到库函数区之后程序权限受到正确的限制。
- **流程**：用户应用程序的主函数中设置库函数数据区上下界寄存器组，跳转时可以使用正常的跳转指令到库函数区，同时硬件设置主函数返回地址寄存器。对于自由跳转区边界寄存器，可以在U态主函数区或者S态主函数区进行初始化，由于ucas-os目前将所有需要嵌套调用的子函数视为自由跳转区，因此把它们放置在同一个代码段之后，在S态主函数区进行初始化。

3、 库函数返回主函数

- **含义**：需要保证控制流的正常，即库函数返回主函数时，返回地址必须和之前跳进库函数区的地址相同。另外，需要特殊区分库函数进行主函数调用的情况。
- **流程**：保存跳转地址到库函数返回地址寄存器（DasicsReturnPC寄存器）。硬件检查跳转地址是否等于主函数返回地址寄存器或主函数调用入口寄存器（DasicsMaincallEntry寄存器），检查失败触发用户态中断。如果等于主函数返回地址寄存器，则继续执行主函数；如果等于主函数调用入口，则执行主函数调用流程。

4、 库函数进行主函数调用

- **含义**：支持主函数调用功能，对库函数区的一些权限进行修改。
- **流程**：主函数根据调用参数进行处理，可能修改库函数权限寄存器和库函数数据区上下界寄存器以及主函数数据区。主函数入口函数先保存DasicsReturnPC和DasicsFreeZoneReturnPC的值，之后完成调用后恢复两个寄存器的值，再利用PULPRET指令返回，保证主函数返回地址寄存器上下文在主函数调用前后一致。

5、 用户态中断

- **含义**：对于权限出现错误，需要进行系统处理的时候，使用用户态中断而非系统中断提高程序性能。
- **流程**：有6种中断可能被触发：U态/S态跳转错误、U态/S态读错误、U态/S态写错误，根据代理寄存器配置情况决定是否代理给用户态处理。此外，安全寄存器访问错误按照RV规范，视为illegalInstr例外处理；系统调用检查错误计划在移植yzh师兄的工作之后实现。用户态中断返回使用URET指令，返回到uepc寄存器所指的位置。

6、 库函数调用库函数（自由跳转区）

- **含义**：不同的库函数之间可能存在调用，需要设置库函数跳转到库函数的自由跳转区来允许嵌套调用。
- **流程**：设置自由跳转区之后，对于库函数之间的跳转情况，可以分为如下四种类型进行处理：
  - 自由跳转区-> 自由跳转区：允许，硬件不会做额外的操作
  - 非自由跳转区 -> 自由跳转区：允许，硬件会记录返回非自由跳转区的返回地址(DasicsFreeZoneReturnPC)
  - 自由跳转区 -> 非自由跳转区：硬件将会检查跳转的目标地址是否等于DasicsFreeZoneReturnPC，如果不等的话，将会引发DASICS跳转例外
  - 非自由跳转区 -> 非自由跳转区：不允许，硬件会直接引发DASICS跳转例外

7. 对内核的DASICS保护

- **含义**：将内核的某些部分（例如设备驱动，文件系统）通过DASICS机制保护，避免第三方驱动或内核模块对内核的攻击。
- **实现的功能与处理流程**：
  - 内核启动时对部分代码功能（例如设备驱动，文件系统，插入的内核模块）进行分区隔离，设置为库函数区。内核单独有一套主函数代码区配置&边界寄存器，在切换用户态和内核态时，硬件自动切换使用对应的配置&边界寄存器。库函数代码区配置&边界寄存器则是内核和用户复用，保存/恢复程序上下文的时候把这些寄存器切给内核/用户使用。主函数代码区配置寄存器暂时与用户态分开，但如果后续两个寄存器都不需要使用太多的reserved bits的话，可以将其合并在一起。
  - 修改例外处理流程，在进入系统调用或者中断等例外的时候硬件切到内核的边界寄存器，这一点通过内核/用户各使用一套主函数边界寄存器的方法得到了解决。
  - 内核中触发权限错误需要触发中断，现有的scause寄存器需要增加新的对应值。这一点通过引入新的6种DASICS例外得到了解决。
  - 如果用户态不想处理中断，允许其转交给内核处理。目前的实现是：用户态中断有默认的处理函数，该函数里面调syscall交给内核处理。

五、待实现功能

1. 库函数调用库函数（权限变化）

- **含义**：不同的库函数之间可能存在调用，如果两个库函数的权限不同，则需要保护跳转的同时权限也进行权限的切换。
- **流程**：利用主函数调用入口寄存器进入switch_pulp函数（用堆栈传参），该函数完成切换权限的操作，返回的时候调用switch_pulp函数返回。两次switch_pulp函数的调用通过参数区分修改的权限内容（参数mode为0代表进入准备调用另一个库函数，为1代表返回）。

2、对内核的PULP保护（动态分配&PMP）

- **含义**：将内核的某些部分（例如设备驱动，文件系统）通过PULP机制保护，避免第三方驱动或内核模块对内核的攻击。
- **需要实现的功能和流程**：
    - 设置对应的库函数区权限（比如设备驱动只能访问对应设备的内存地址），地址设置上考虑如下两个设计：
        - 允许内核中的库函数区，动态申请buffer，但是需要先给每个区域分配好一个堆空间，每次从这个堆空间中申请buffer。***（目前的实现中，边界寄存器的数目有限，难以做到细粒度的保护，只能对整个堆进行保护，因此需要新的解决方案，如PLB）***
        - 由于DMA必须使用物理地址作传输地址，保护时考虑软件结合PMP限制。***（由于目前NutShell没有实现PMP机制，因此可以考虑移植香山的PMP，或者暂时不管等后续项目整体迁移到香山上再做）***

3. 移植杨宗昊的毕设，启动Linux

4. 用户态页表