1

TLB 工作原理

TLB 是处理器内存管理单元中的硬件部件将虚存页地址转换为物理内存地址的一个加速，即为这个地址映射提供了转换缓冲。

TLB 工作时，首先根据虚拟地址在其中查询，如果查询直接 Hit 到物理地址，便可根据物理地址查询 Cache 或者 Main Memory，否则转译虚拟地址为物理地址，进行查询，并将这一次查询的结果（虚拟地址与物理地址对）按照一定策略加入 TLB 作为 PTE。

MIPS32 通过 ITLB，JTLB，DTLB 三个部分来组成 MMU 的结构，其中 ITLB 处理指令虚拟地址，DTLB 处理数据虚拟地址，如果 ITLB 和 DTLB 中虚拟地址缺失，则交由 JTLB 处理，如果 JTLB 成功查询到，则将此表项返回复制给 ITLB 或 DTLB。

Page fault 异常与 TLB 缺失异常的区别：

Page fault 异常是指当前待查询的虚存地址在物理内存中没有页表项对应，对其处理则是从外部存储读入所需的数据。

TLB 缺失异常则是 TLB 中的 PTE 不存在待查询的虚拟地址，此时需要从内存页表中读取虚拟地址对应的物理地址，并填入 TLB 中，此时的虚拟地址可能在物理内存中仍有页表项的对应，不一定会触发 Page fault 异常，即便产生也与 TLB 缺失异常分为两步发生。

2

2.1

保存当前栈顶位置的数据即函数的返回地址。

2.2

计算并保存当前运行状态的栈顶位置（跳过返回地址，方便恢复时 `push`）。

2.3

恢复指针 `i` 指向内存中的运行状态时其返回地址，通过 `pushq` 使其成为本函数的返回地址。

2.4

至少需要 $14 \times 64\text{bit}/(8\text{Byte}/\text{bit}) = 112\text{Byte}$。

2.5

`r10` 与 `r11` 是 `Caller Saved` 即被调用者负责保存，调用 `SwapContent` 函数时，应当已经保存，恢复运行状态后，也应当自行恢复。

而 `rax` 是作为函数的返回值，也就是 `SwapContent` 函数的返回值。

3

3.1

`result` 通过 `movslq %esi, %rdx` 存储在寄存器 `%rdx` 中，并且最后交给 `%rax` 返回。

3.2

`i` 存储在 `%edi` 中，换而言之，与函数第二个参数 `b` 共用同一个寄存器表示。

3.3

```
EXPR1 = a
EXPR2 = b
EXPR3 = 0
EXPR4 = a
EXPR5 = i
```

4

指令 `lw $2, 65540($4)` 替换为 `ll $2, 65540($4)`，指令 `sw $2, 65540($4)` 替换为 `sc $2, 65540($4)`。

填空如下：

```
atomic_inc:
    lui $1, 1
    addu $1, $1, $4
    lw $2, 4($1)
    addiu $2, $2, 1
```

5

最大规格化数的 `exp` 是 30，`frac` 是 $(1111111111)_2$。

表示数值是 $(1111111111100000)_2 = 65504$。

`s` 的值为 $(0111101111111111)_2 = 31743$。

6

对应表项分别为 `0x8E2AE/F`，`0x8E2AE/F`，`0x8E2BC/D`，`0x8E2BC/D`。

6.1

miss

6.2

0x22340324

6.3

0x32345330

6.4

miss