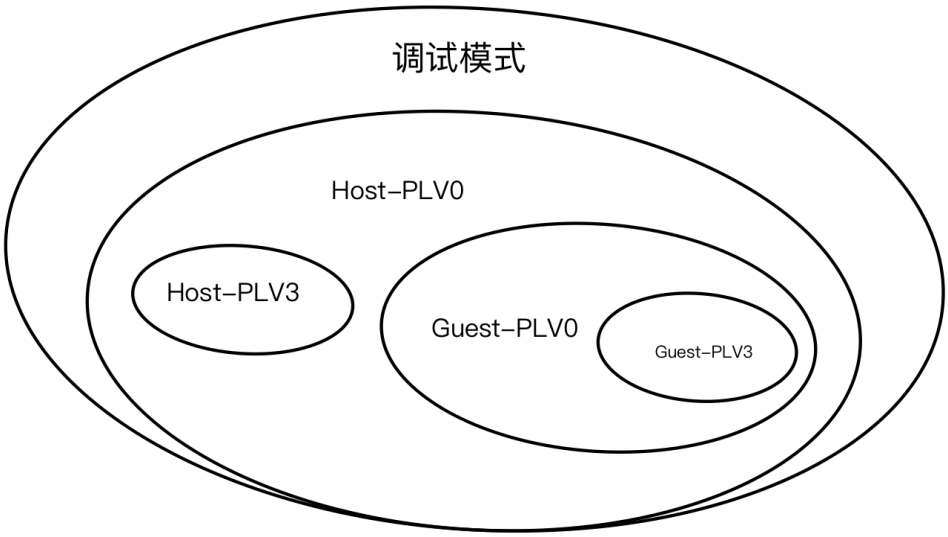
# CA_Assignment4

| | |
|---|---|
| Assign | |
| Property | |
| tag | homework |
| 姓名 | 周鹏宇 |
| 学号 | 2019K8009929039 |

## 2.1

请以一种指令系统为例，说明其定义了哪些运行级别，以及这些运行级别之间的区别与联系。




LoongArch不同运行级别的关系图

- 运行级别
  - 调试模式（Debug Mode）
  - 主机模式（Host Mode）
    - PLV0-PLV3四个权限等级
  - 客机模式（Guest Mode）
    - PLV0-PLV3四个权限等级

- 联系
  - 处理器某一时刻只能存在于某一种运行级别中
  - 处理器上电复位后处于Host-PLV0级，随后根据需要在不同级之间转换
- 区别
  - 不同运行级别可访问并控制的处理器资源不同，关系如上图所示

## 2.2

请用 C 语言伪代码形式描述一个采用段页式存储管理机制的计算机系统虚实地址转换的过程

```
struct Vir_addr{
    segment;
    virPage;
    offset;
}Vir_addr;

Page_pffset = segment[Vir_addr.segment];
Phys_tag = page[Vir_addr.Page_tag] << Page_pffset;
phys_addr = Phys_tag << Phys_tag;
```

## 2.3

请简述桌面电脑 PPT 翻页过程中用户态和核心态的转换过程

当按下键盘后，处理器接受到中断信号，从用户态切换到核心态以响应中断，随后回到用户态运行 PowerPoint；PowerPoint 调用显示驱动程序，处理器进入核心态以执行显示操作，随后回到用户态。

## 2.4

给定下列程序片段：

```
A = B + C
B = A + C
C = B + A
```

（1）写出上述程序片段在四种指令系统类型（堆栈型、累加器型、寄存器-存储器型、寄存器-寄存器型）中的指令序列。

```
PUSH B
PUSH C
ADD
POP  A
PUSH A
PUSH C
ADD
POP  B
PUSH B
PUSH A
ADD
POP  C
```

```
LOAD  B
ADD   C
STORE A
ADD   C
STORE B
ADD   A
STORE C
```

```
LOAD   R1,B
ADD    R1,C
STORE  A, R1
ADD    R1,C
STORE  B, R1
ADD    R1,A
STORE  C, R1
```

```
LOAD  R1,B
LOAD  R2,C
ADD   R3,R2,R1
STORE A, R3
ADD   R1,R2,R3
STORE B, R1
ADD   R2,R3,R1
STORE C,R2
```

（2）假设四种指令系统类型都属于 CISC 型，令指令码宽度为 $x$ 位，寄存器操作数宽度为 $y$ 位，内存地址操作数宽度为 $z$ 位，数据宽度为 $w$ 位。分析指令的总位数和所有内存访问的总位数

- 堆栈型：
  - 总位数：$9(x+z)+3x=12x+9z$

        - 访存指令总位数：$9x + 9z$
        - 访问数据总位数：$9w$
    - 累加器型
        - 总位数：$4(x + z) + 3x = 7x + 4z$
        - 访存指令总位数：$7x + 4z$
        - 访问数据总位数：$7w$
    - 寄存器-存储器型
        - 总位数: $7(x + y + z) = 7x + 7y + 7z$
        - 访存指令总位数: $7x + 7y + 7z$
        - 访问数据总位数: $7w$
    - 寄存器-寄存器型
        - 总位数：$5(x + y + z) + 3(x + 3y) = 8x + 14y + 5z$
        - 访存指令总位数：$5x + 5y + 5z$
        - 访存数据总位数：$5w$

（3）微处理器由 32 位时代进入了 64 位时代，上述四种指令系统类型哪种更好?

由（2）中的计算可以注意到，寄存器-寄存器型是四者中访存数据总位数（访存次数）最少的，在64位的情况下，这一优势会被放大。此外，寄存器-寄存器型更容易判断指令冲突的发生（如RAW问题），故寄存器-寄存器型最好

---

5. 写出 0xDEADBEEF 在大尾端和小尾端下在内存中的排列(由地址 0 开始)

---

- 小端序：EF BE AD DE
- 大端序：DE AD BE EF

---

6. 在你的机器上编写 C 程序来得到不同数据类型的字节数，给出程序和结果。

```
#include <stdio.h>

typedef struct {
    char c;
    short s;
    double d;
}struct_t;
```

```
typedef union {
    char c;
    int i;
}union_t;
int main() {
    char c;
    short s;
    int i;
    long l;
    long long ll;
    float f;
    double d;
    long double ld;
    void * v;
    struct_t str;
    union_t uni;
    printf("%lu\n", sizeof (c));
    printf("%lu\n", sizeof (s));
    printf("%lu\n", sizeof (i));
    printf("%lu\n", sizeof (l));
    printf("%lu\n", sizeof (ll));
    printf("%lu\n", sizeof (f));
    printf("%lu\n", sizeof (d));
    printf("%lu\n", sizeof (ld));
    printf("%lu\n", sizeof (v));
    printf("%lu\n", sizeof (str));
    printf("%lu\n", sizeof (uni));

    return 0;
}
```

```
master ●✚  ./a.out
1
2
4
8
8
4
8
16
8
16
4
```

7. 根据 LoongArch 指令集的编码格式设计 2RI16、1RI21 和 I26 三种编码格式的直接转移指令各自的跳转范围。

记读寄存器的内容为r，此时的PC值为p，则

- 2RI16指令：跳转范围为: $p \pm 2^{17}$,之所以是17是需要考虑指令编码时记录在立即数字段的偏移地址末两位0

- 1RI21指令：
  - 读寄存器：$p \pm 2^{22}$
  - 写寄存器：$PC \pm 2^{22}$
  - 绝对跳转：$2^{23}$
- I26指令：
  - 读寄存器：$p \pm 2^{27}$
  - 绝对跳转：$2^{28}$

---

8. 仅使用对齐访存指令写出如图 2.9 所示(图略)的不对齐加载(小尾端)。

```
ld.w    $t1, $zero, 0
ld.w    $t2, $zero, 4
srli.w  $t1, $t1, 8
slli.w  $t2, $t2, 24
or.w    $t0, $t1, $t2
```