B62008H 计算机体系结构 2017 年 9 月 18 日

作业一

主讲教师：胡伟武 苏孟豪 张远航 2015K8009929045

第一章

1. 计算机系统可划分为哪几个层次，各层次之间的界面是什么？你认为这样划分的意义何在？

   计算机系统可分成应用程序、操作系统、硬件系统、晶体管四个大的层次；它们之间的界面分别是应用程序编程接口（API）、指令系统（ISA）和工艺模型。

   这样可以让不同层次的开发者利用接口，关注相应的层次而无需过分关注该层以下的细节；此外，统一的指令系统使得在不同的机器上运行相同的程序成为可能。

2. 在三台不同指令系统的计算机上运行同一程序 P 时，A 机器需要执行 1.0×10^9 条指令，B 机器需要执行 2.0×10^9 条指令，C 机器需要执行 3.0×10^9 条指令，但三台机器的实际执行时间都是 100 秒。请分别计算出这 3 台机器的 MIPS，并指出运行程序 P 时哪台机器的性能最高。

   A 机器：$1.0 \times 10^9/(100 \times 10^6) = 10\text{MIPS}$

   B 机器：$2.0 \times 10^9/(100 \times 10^6) = 20\text{MIPS}$

   C 机器：$3.0 \times 10^9/(100 \times 10^6) = 30\text{MIPS}$

   从执行程序所用的时间来看，三台机器的性能一样高。

3. 假设某程序中可向量化的百分比为 P，现在给处理器中增加向量部件以提升性能，向量部件的加速比为 S。请问增加向量部件后的处理器运行该程序的性能提升幅度是多少？

   按 Amdahl 定律，性能提升幅度为 $\dfrac{1}{(1-P)+P/S}$。

4. 处理器的功耗可简单分为静态功耗和动态功耗两部分，其中静态功耗的特性符合欧姆定律，动态功耗在其他条件相同的情况下与频率成正比。现对某处理器进行功耗测试，得到如下数据：关闭时钟，电压 1.0V 时，电流为 100mA；时钟频率为 1GHz，电压 1.1V 时，电流为 2100mA。请计算在时钟频率为 2GHz，电压 1.1V 时，此处理器的总功耗。

   处理器等效电阻为 1.0V/100mA=10Ω，1.1V 时，静态电流为 1.1V/10Ω=110mA，故 1GHz 时动态电流为 2100mA-110mA=1990mA；由于动态功耗与频率成正比，2GHz 时动态电流为 1990mA×2GHz/1GHz=3980mA，故总功耗为 (3980+110)× 1.1W=4.499W。

5. 在一台个人计算机上进行 SPEC CPU 2000 的测试，分别给出无编译优化选项和编译优化选项为-O2 的单核测试报告。

无优化的结果：

```
2017年 09月 18日 星期一 13:22:08 CST
[CPU]:
 Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
 Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
 Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
 Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
 Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
 Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
 Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
 Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
[Memory]:
       16117544 kB
[Linux]:
Linux caszhang 4.8.0-42-generic #45~16.04.1-Ubuntu SMP Thu Mar 9 14:10:58 UTC 2017
[MAC]:
[Compiler]:
gcc (Ubuntu 5.4.1-2ubuntu1~16.04) 5.4.1 20160904
g++ (Ubuntu 5.4.1-2ubuntu1~16.04) 5.4.1 20160904
GNU Fortran (Ubuntu 5.4.1-2ubuntu1~16.04) 5.4.1 20160904
[Result]:
 Success 164.gzip ratio=1295.13, runtime=108.097170
 Success 175.vpr ratio=1650.23, runtime=84.836886
 Success 181.mcf ratio=2707.18, runtime=66.489955
 Success 186.crafty ratio=3003.29, runtime=33.296792
 Success 197.parser ratio=1310.67, runtime=137.334629
 Success 252.eon ratio=667.00, runtime=194.902906
 Success 253.perlbmk ratio=3375.30, runtime=53.328606
 Success 254.gap ratio=4166.04, runtime=26.403973
 Success 255.vortex ratio=2868.41, runtime=66.238891
 Success 256.bzip2 ratio=1430.06, runtime=104.890460
 Success 168.wupwise ratio=1996.47, runtime=80.141337
 Success 171.swim ratio=2759.06, runtime=112.357272
 Success 172.mgrid ratio=599.95, runtime=300.022522
 Success 173.applu ratio=1034.49, runtime=202.997670
 Success 177.mesa ratio=2650.69, runtime=52.816410
 Success 178.galgel ratio=2577.74, runtime=112.501660
 Success 179.art ratio=6785.45, runtime=38.317269
```

```
 Success 183.equake ratio=2682.26, runtime=48.466632
 Success 187.facerec ratio=2961.25, runtime=64.162062
 Success 188.ammp ratio=1743.60, runtime=126.175410
 Success 189.lucas ratio=3500.50, runtime=57.134683
 Success 191.fma3d ratio=2289.44, runtime=91.725641
 Success 200.sixtrack ratio=599.83, runtime=183.384448
 Success 301.apsi ratio=1586.55, runtime=163.877843
```

开启 O2 优化后的结果：

```
2017年 09月 18日 星期一 12:30:48 CST
[CPU]:
 Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
 Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
 Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
 Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
 Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
 Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
 Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
 Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
[Memory]:
       16117544 kB
[Linux]:
Linux caszhang 4.8.0-42-generic #45~16.04.1-Ubuntu SMP Thu Mar 9
14:10:58 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux
[MAC]:
[Compiler]:
gcc (Ubuntu 5.4.1-2ubuntu1~16.04) 5.4.1 20160904
g++ (Ubuntu 5.4.1-2ubuntu1~16.04) 5.4.1 20160904
GNU Fortran (Ubuntu 5.4.1-2ubuntu1~16.04) 5.4.1 20160904
[Result]:
 Success 164.gzip ratio=2220.28, runtime=63.054973
 Success 175.vpr ratio=3247.96, runtime=43.104039
 Success 176.gcc ratio=5395.87, runtime=20.385954
 Success 181.mcf ratio=3867.93, runtime=46.536487
 Success 186.crafty ratio=4557.63, runtime=21.941217
 Success 197.parser ratio=2682.08, runtime=67.111997
 Success 252.eon ratio=5500.37, runtime=23.634791
 Success 254.gap ratio=4340.22, runtime=25.344313
```

```
Success 255.vortex ratio=5415.74, runtime=35.082926
Success 256.bzip2 ratio=3076.00, runtime=48.764682
Success 300.twolf ratio=4269.12, runtime=70.272067
Success 168.wupwise ratio=5087.14, runtime=31.451856
Success 171.swim ratio=9462.89, runtime=32.759550
Success 172.mgrid ratio=3575.35, runtime=50.344672
Success 173.applu ratio=5743.34, runtime=36.564118
Success 177.mesa ratio=5403.92, runtime=25.907121
Success 178.galgel ratio=10170.66, runtime=28.513398
Success 179.art ratio=14968.57, runtime=17.369734
Success 183.equake ratio=8743.82, runtime=14.867641
Success 187.facerec ratio=5985.64, runtime=31.742612
Success 188.ammp ratio=4386.29, runtime=50.156318
Success 189.lucas ratio=7933.52, runtime=25.209506
Success 191.fma3d ratio=5436.37, runtime=38.628701
Success 200.sixtrack ratio=1839.29, runtime=59.805706
Success 301.apsi ratio=5604.85, runtime=46.388394
```

第二章

1. 列出一种指令系统的不同运行级别之间的关系。

   首先要对教材这一部分的用词做一点勘误：**运行级别**（runlevel）指的是 Unix 或者 Linux 等类 Unix 操作系统下不同的运行模式，指令系统（处理器）对应的名词应该是特权级别（privilege level）和运行（工作）模式。

   MIPS 的关系在教材中已经说的比较清楚，这里以 ARM 为例。ARM 处理器共支持 7 种特权模式：用户模式（usr），正常的程序执行状态；快速中断模式（fiq），用于高速数据传输或通道处理；外部中断模式（irq），用于通用的中断处理；管理模式（svc），操作系统使用的保护模式；数据访问终止模式 (abt)，当数据或指令预取终止时进入该模式，可用于虚拟存储及存储保护；系统模式（sys），运行具有特权的操作系统任务；未定义指令中止模式（und），未定义的指令执行时进入该模式，可用于支持硬件协处理器的软件仿真。

   ARM 处理器的运行模式可以通过软件、外部中断或异常处理改变。大多数的应用程序运行在用户模式下，当处理器运行在用户模式下时，某些被保护的系统资源是不能被访问的。笼统来说，ARM 有两个特权级别：用户模式和特权模式。以上七种模式中，除用户模式以外，其余的所有 6 种模式称之为非用户模式，或特权模式。

2. 用 C 语言描述段页式存储管理的地址转换过程。

```c
#define N_SEG_ENTRIES           1024
```

```
#define N_PAGE_TABLE_ENTRIES    1024

typedef struct{
    int segn;   // segment number
    int vpn;    // virtual page number
    int offset; // offset in page
} vaddr;

typedef struct{
    int ppn;    // physical page number
    int offset; // offset in page
} paddr;

typedef struct{
    int segl;   // segment length
    int base;   // segment base address
} seg_entry;

typedef struct{
    bool valid; // valid–invalid bit
    int ppn;    // physical page number
} page_entry;

seg_entry seg_table[N_SEG_ENTRIES];
page_entry page_table[N_SEG_ENTRIES][N_PAGE_TABLE_ENTRIES];

paddr MMU_translate(vaddr va) {
    paddr pa;

    // check if segment base address is valid
    if (va.segn > N_SEG_ENTRIES) {
        printf("ERROR segment does not exist");
        error_handler();
    }
    // check translation failure
    if (va.vpn > seg_table[va.seg].segl || !pa.ppage.valid) {
        printf("ERROR segment fault");
        error_handler();
```

```
    }
    else if (!pa.ppage.valid) {
        printf("ERROR segment fault");
        error_handler();
    }
    else {
        pa.ppage = page_table[seg_table[va.segn].base][va.vpn];
        pa.offset = va.offset;

        return pa;
    }
}
```

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。

   PowerPoint 等待外部输入的过程中，系统处于用户态；按下键盘后，需要进行中断处理，转换至核心态，完成后返回到用户态，PowerPoint 准备下一页要显示的内容；接下来，为了调用显示驱动程序把显示的内容送到显存，再次转换至核心态，屏幕刷新后返回到 PowerPoint 便重新转换至用户态。