CA_Assignment6

Assign	
tag	
姓名	周鹏宇
学号	2019K8009929039

1. 列出范例程序的参数传递过程

共有以下参数：char, int, short, long, float, double, struct small, struct big

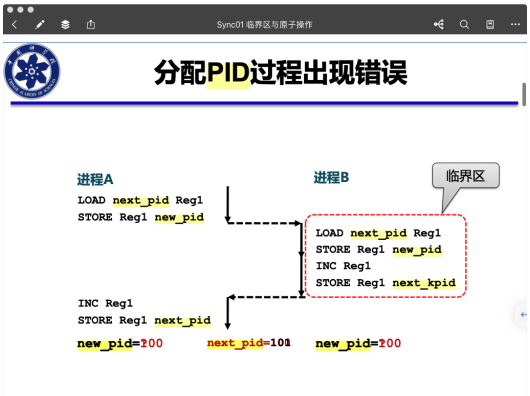
- char, int, long, small, short都在64位以内，故可用单个寄存器存储
- float和double类型存在单个浮点参数寄存器中进行传递
- big是32个byte大小，通过引用传递参数

故有：

参数序号	传递方式	内容
0	$a0	char 0x61
1	$a1	short 0xffff
2	$a2	int 1
3	$a4	long 2
4	$fa0	float 3.0
5	$fa1	double 4.0
6	$a5	0-15位为char，32-63位为int
7	$a6	结构体big的地址
8	$a7	long 9

2.

a. 用 LoongArch 汇编程序来举例并分析
未同步的线程之间进行共享数据访问出
错的情况
本题目参考操作系统课程给出的例子：



将该段以loongarch形式翻译出后为：

```
*不妨记pid被存在寄存器s0所指内存中，则
ld.w   $t0, s0 , 0 *读pid
addi.w $t0, $t0, 1 *pid++
sw.w   $t0, s0 , 0 *写回pid
```

另一段程序与之基本相同，当两者运行且未同步时，会发生如上图所示的冲突

b. 用 LL/SC 指令改写你的程序，使它们的共享数据访问正确。

```
ll.w   $t0, s0 , 0 *读pid
addi.w $t0, $t0, 1 *pid++
sc.w   $t0, s0 , 0 *写回pid
```

3.

a. 写一段冒泡排序的C程序，在你的机器上安装 LoongArch 交叉编译器，通过编译-反汇编的方式提取函数调用的核心片段。

```
#include<stdio.h>
#include<stdlib.h>
#define N 6
void bubble_sort(int a[],int n);
void bubble_sort(int a[],int n)
{
    for(int i=0; i<n-1; i++)
    {
        for(int j=0; j<n-1-i; j++)
        {
            if(a[j] > a[j+1])
            {
                int temp = a[j];
                a[j] = a[j+1];
                a[j+1]=temp;
            }
        }
    }
}
```

```
int  main()
{
    int num[N] = {1, 1, 4, 5, 1, 4};
    bubble_sort(num, N);
    for(int i=0; i<N; i++){
        printf("%d  ", num[i]);
        printf("\n");
    }

    return 0;
}
```

```
000101f0 <bubble_sort>:
   101f0: 02bf4063  addi.w  $r3,$r3,-48(0xfd0)
   101f4: 2980b076  st.w  $r22,$r3,44(0x2c)
   101f8: 0280c076  addi.w  $r22,$r3,48(0x30)
   101fc: 29bf72c4  st.w  $r4,$r22,-36(0xfdc)
   10200: 29bf62c5  st.w  $r5,$r22,-40(0xfd8)
   10204: 29bfb2c0  st.w  $r0,$r22,-20(0xfec)
   10208: 5000cc00  b 204(0xcc) # 102d4 <bubble_sort+0xe4>
   1020c: 29bfa2c0  st.w  $r0,$r22,-24(0xfe8)
   10210: 5000a000  b 160(0xa0) # 102b0 <bubble_sort+0xc0>
   10214: 28bfa2c9  ld.w  $r9,$r22,-24(0xfe8)
   10218: 00408929  slli.w  $r9,$r9,0x2
   1021c: 28bf72c8  ld.w  $r8,$r22,-36(0xfdc)
   10220: 00102509  add.w $r9,$r8,$r9
   10224: 28800129  ld.w  $r9,$r9,0
   10228: 28bfa2c8  ld.w  $r8,$r22,-24(0xfe8)
   1022c: 02800508  addi.w  $r8,$r8,1(0x1)
   10230: 00408908  slli.w  $r8,$r8,0x2
   10234: 28bf72c7  ld.w  $r7,$r22,-36(0xfdc)
   10238: 001020e8  add.w $r8,$r7,$r8
   1023c: 28800108  ld.w  $r8,$r8,0
   10240: 64006509  bge $r8,$r9,100(0x64) # 102a4 <bubble_sort+0xb4>
   10244: 28bfa2c9  ld.w  $r9,$r22,-24(0xfe8)
   10248: 00408929  slli.w  $r9,$r9,0x2
   1024c: 28bf72c8  ld.w  $r8,$r22,-36(0xfdc)
   10250: 00102509  add.w $r9,$r8,$r9
   10254: 28800129  ld.w  $r9,$r9,0
   10258: 29bf92c9  st.w  $r9,$r22,-28(0xfe4)
   1025c: 28bfa2c9  ld.w  $r9,$r22,-24(0xfe8)
   10260: 02800529  addi.w  $r9,$r9,1(0x1)
   10264: 00408929  slli.w  $r9,$r9,0x2
   10268: 28bf72c8  ld.w  $r8,$r22,-36(0xfdc)
   1026c: 00102508  add.w $r8,$r8,$r9
   10270: 28bfa2c9  ld.w  $r9,$r22,-24(0xfe8)
   10274: 00408929  slli.w  $r9,$r9,0x2
   10278: 28bf72c7  ld.w  $r7,$r22,-36(0xfdc)
   1027c: 001024e9  add.w $r9,$r7,$r9
   10280: 28800108  ld.w  $r8,$r8,0
   10284: 29800128  st.w  $r8,$r9,0
   10288: 28bfa2c9  ld.w  $r9,$r22,-24(0xfe8)
   1028c: 02800529  addi.w  $r9,$r9,1(0x1)
   10290: 00408929  slli.w  $r9,$r9,0x2
   10294: 28bf72c8  ld.w  $r8,$r22,-36(0xfdc)
   10298: 00102509  add.w $r9,$r8,$r9
   1029c: 28bf92c8  ld.w  $r8,$r22,-28(0xfe4)
   102a0: 29800128  st.w  $r8,$r9,0
   102a4: 28bfa2c9  ld.w  $r9,$r22,-24(0xfe8)
   102a8: 02800529  addi.w  $r9,$r9,1(0x1)
   102ac: 29bfa2c9  st.w  $r9,$r22,-24(0xfe8)
   102b0: 28bf62c9  ld.w  $r9,$r22,-40(0xfd8)
   102b4: 02bffd28  addi.w  $r8,$r9,-1(0xfff)
```

```
   102b8: 28bfb2c9  ld.w  $r9,$r22,-20(0xfec)
   102bc: 00112509  sub.w $r9,$r8,$r9
   102c0: 28bfa2c8  ld.w  $r8,$r22,-24(0xfe8)
   102c4: 63ff5109  blt $r8,$r9,-176(0x3ff50) # 10214 <bubble_sort+0x24>
   102c8: 28bfb2c9  ld.w  $r9,$r22,-20(0xfec)
   102cc: 02800529  addi.w  $r9,$r9,1(0x1)
   102d0: 29bfb2c9  st.w  $r9,$r22,-20(0xfec)
   102d4: 28bf62c9  ld.w  $r9,$r22,-40(0xfd8)
   102d8: 02bffd29  addi.w  $r9,$r9,-1(0xfff)
   102dc: 28bfb2c8  ld.w  $r8,$r22,-20(0xfec)
   102e0: 63ff2d09  blt $r8,$r9,-212(0x3ff2c) # 1020c <bubble_sort+0x1c>
   102e4: 03400000  andi  $r0,$r0,0x0
   102e8: 2880b076  ld.w  $r22,$r3,44(0x2c)
   102ec: 0280c063  addi.w  $r3,$r3,48(0x30)
   102f0: 4c000020  jirl  $r0,$r1,0

000102f4 <main>:
   102f4: 02bf4063  addi.w  $r3,$r3,-48(0xfd0)
   102f8: 2980b061  st.w  $r1,$r3,44(0x2c)
   102fc: 2980a076  st.w  $r22,$r3,40(0x28)
   10300: 0280c076  addi.w  $r22,$r3,48(0x30)
   10304: 1c000289  pcaddu12i $r9,20(0x14)
   10308: 029fb129  addi.w  $r9,$r9,2028(0x7ec)
   1030c: 28800124  ld.w  $r4,$r9,0
   10310: 28801125  ld.w  $r5,$r9,4(0x4)
   10314: 28802126  ld.w  $r6,$r9,8(0x8)
   10318: 28803127  ld.w  $r7,$r9,12(0xc)
   1031c: 28804128  ld.w  $r8,$r9,16(0x10)
   10320: 28805129  ld.w  $r9,$r9,20(0x14)
   10324: 29bf52c4  st.w  $r4,$r22,-44(0xfd4)
   10328: 29bf62c5  st.w  $r5,$r22,-40(0xfd8)
   1032c: 29bf72c6  st.w  $r6,$r22,-36(0xfdc)
   10330: 29bf82c7  st.w  $r7,$r22,-32(0xfe0)
   10334: 29bf92c8  st.w  $r8,$r22,-28(0xfe4)
   10338: 29bfa2c9  st.w  $r9,$r22,-24(0xfe8)
   1033c: 02bf52c9  addi.w  $r9,$r22,-44(0xfd4)
   10340: 02801805  addi.w  $r5,$r0,6(0x6)
   10344: 00150124  move  $r4,$r9
   10348: 57feabff  bl  -344(0xffffea8) # 101f0 <bubble_sort>
   1034c: 29bfb2c0  st.w  $r0,$r22,-20(0xfec)
   10350: 50003c00  b 60(0x3c) # 1038c <main+0x98>
   10354: 28bfb2c9  ld.w  $r9,$r22,-20(0xfec)
   10358: 00408929  slli.w  $r9,$r9,0x2
   1035c: 02bfc2c8  addi.w  $r8,$r22,-16(0xff0)
   10360: 00102509  add.w $r9,$r8,$r9
   10364: 28bf9129  ld.w  $r9,$r9,-28(0xfe4)
   10368: 00150125  move  $r5,$r9
   1036c: 1c000284  pcaddu12i $r4,20(0x14)
   10370: 029df084  addi.w  $r4,$r4,1916(0x77c)
   10374: 5400b800  bl  184(0xb8) # 1042c <printf>
   10378: 02802804  addi.w  $r4,$r0,10(0xa)
   1037c: 54011000  bl  272(0x110) # 1048c <putchar>
   10380: 28bfb2c9  ld.w  $r9,$r22,-20(0xfec)
   10384: 02800529  addi.w  $r9,$r9,1(0x1)
   10388: 29bfb2c9  st.w  $r9,$r22,-20(0xfec)
   1038c: 28bfb2c9  ld.w  $r9,$r22,-20(0xfec)
   10390: 02801408  addi.w  $r8,$r0,5(0x5)
   10394: 67ffc109  bge $r8,$r9,-64(0x3ffc0) # 10354 <main+0x60>
   10398: 00150009  move  $r9,$r0
   1039c: 00150124  move  $r4,$r9
   103a0: 2880b061  ld.w  $r1,$r3,44(0x2c)
   103a4: 2880a076  ld.w  $r22,$r3,40(0x28)
   103a8: 0280c063  addi.w  $r3,$r3,48(0x30)
   103ac: 4c000020  jirl  $r0,$r1,0
```

b. 改变编译的优化选项，记录函数调用核心片段的变化，并分析不同优化选项的效果。

```
000101f0 <bubble_sort>:
   101f0: 02bffca5  addi.w  $r5,$r5,-1(0xfff)
   101f4: 64005005  bge $r0,$r5,80(0x50) # 10244 <bubble_sort+0x54>
   101f8: 001500aa  move  $r10,$r5
   101fc: 0015008b  move  $r11,$r4
   10200: 004088a6  slli.w  $r6,$r5,0x2
   10204: 00101886  add.w $r6,$r4,$r6
   10208: 50003000  b 48(0x30) # 10238 <bubble_sort+0x48>
   1020c: 02801129  addi.w  $r9,$r9,4(0x4)
   10210: 58001d26  beq $r9,$r6,28(0x1c) # 1022c <bubble_sort+0x3c>
   10214: 28800128  ld.w  $r8,$r9,0
   10218: 28801127  ld.w  $r7,$r9,4(0x4)
   1021c: 67fff0e8  bge $r7,$r8,-16(0x3fff0) # 1020c <bubble_sort+0x1c>
   10220: 29800127  st.w  $r7,$r9,0
   10224: 29801128  st.w  $r8,$r9,4(0x4)
   10228: 53ffe7ff  b -28(0xfffffe4) # 1020c <bubble_sort+0x1c>
   1022c: 02bffd4a  addi.w  $r10,$r10,-1(0xfff)
   10230: 02bff0c6  addi.w  $r6,$r6,-4(0xffc)
   10234: 58001140  beq $r10,$r0,16(0x10) # 10244 <bubble_sort+0x54>
   10238: 00150169  move  $r9,$r11
   1023c: 63ffd80a  blt $r0,$r10,-40(0x3ffd8) # 10214 <bubble_sort+0x24>
   10240: 53ffefff  b -20(0xffffffec) # 1022c <bubble_sort+0x3c>
   10244: 4c000020  jirl  $r0,$r1,0

00010248 <main>:
   10248: 02bf4063  addi.w  $r3,$r3,-48(0xfd0)
   1024c: 2980b061  st.w  $r1,$r3,44(0x2c)
   10250: 2980a076  st.w  $r22,$r3,40(0x28)
   10254: 29809077  st.w  $r23,$r3,36(0x24)
   10258: 29808078  st.w  $r24,$r3,32(0x20)
   1025c: 1c000289  pcaddu12i $r9,20(0x14)
   10260: 029f3129  addi.w  $r9,$r9,1996(0x7cc)
   10264: 28800124  ld.w  $r4,$r9,0
   10268: 28801125  ld.w  $r5,$r9,4(0x4)
   1026c: 28802126  ld.w  $r6,$r9,8(0x8)
   10270: 28803127  ld.w  $r7,$r9,12(0xc)
   10274: 28804128  ld.w  $r8,$r9,16(0x10)
   10278: 28805129  ld.w  $r9,$r9,20(0x14)
   1027c: 29802064  st.w  $r4,$r3,8(0x8)
   10280: 29803065  st.w  $r5,$r3,12(0xc)
   10284: 29804066  st.w  $r6,$r3,16(0x10)
   10288: 29805067  st.w  $r7,$r3,20(0x14)
   1028c: 29806068  st.w  $r8,$r3,24(0x18)
   10290: 29807069  st.w  $r9,$r3,28(0x1c)
   10294: 02801805  addi.w  $r5,$r0,6(0x6)
   10298: 02802064  addi.w  $r4,$r3,8(0x8)
   1029c: 57ff57ff  bl  -172(0xfffff54) # 101f0 <bubble_sort>
   102a0: 02802076  addi.w  $r22,$r3,8(0x8)
   102a4: 02808078  addi.w  $r24,$r3,32(0x20)
   102a8: 1c000297  pcaddu12i $r23,20(0x14)
   102ac: 029de2f7  addi.w  $r23,$r23,1912(0x778)
   102b0: 288002c5  ld.w  $r5,$r22,0
   102b4: 001502e4  move  $r4,$r23
   102b8: 5400ac00  bl  172(0xac) # 10364 <printf>
   102bc: 02802804  addi.w  $r4,$r0,10(0xa)
   102c0: 54010400  bl  260(0x104) # 103c4 <putchar>
   102c4: 028012d6  addi.w  $r22,$r22,4(0x4)
   102c8: 5fffead8  bne $r22,$r24,-24(0x3ffe8) # 102b0 <main+0x68>
   102cc: 00150004  move  $r4,$r0
   102d0: 2880b061  ld.w  $r1,$r3,44(0x2c)
   102d4: 2880a076  ld.w  $r22,$r3,40(0x28)
   102d8: 28809077  ld.w  $r23,$r3,36(0x24)
   102dc: 28808078  ld.w  $r24,$r3,32(0x20)
```

```
   102e0: 0280c063  addi.w  $r3,$r3,48(0x30)
   102e4: 4c000020  jirl  $r0,$r1,0
```

可以注意到在-O1优化后其更加简短，主要体现在减少了对栈的使用（`fdefer-pop`）以及对跳转和循环的优化

```
00010290 <bubble_sort>:
   10290: 02bffca5  addi.w  $r5,$r5,-1(0xfff)
   10294: 64003805  bge $r0,$r5,56(0x38) # 102cc <bubble_sort+0x3c>
   10298: 004088a6  slli.w  $r6,$r5,0x2
   1029c: 00101886  add.w $r6,$r4,$r6
   102a0: 00150089  move  $r9,$r4
   102a4: 28800128  ld.w  $r8,$r9,0
   102a8: 28801127  ld.w  $r7,$r9,4(0x4)
   102ac: 64000ce8  bge $r7,$r8,12(0xc) # 102b8 <bubble_sort+0x28>
   102b0: 29800127  st.w  $r7,$r9,0
   102b4: 29801128  st.w  $r8,$r9,4(0x4)
   102b8: 02801129  addi.w  $r9,$r9,4(0x4)
   102bc: 5fffe8c9  bne $r6,$r9,-24(0x3ffe8) # 102a4 <bubble_sort+0x14>
   102c0: 02bffca5  addi.w  $r5,$r5,-1(0xfff)
   102c4: 02bff0c6  addi.w  $r6,$r6,-4(0xffc)
   102c8: 5fffd8a0  bne $r5,$r0,-40(0x3ffd8) # 102a0 <bubble_sort+0x10>
   102cc: 4c000020  jirl  $r0,$r1,0
```

可以注意到比-O1更加简洁，最典型的就是“消元”更加彻底，调用的寄存器数量更少

```
00010290 <bubble_sort>:
   10290: 02bffca5  addi.w  $r5,$r5,-1(0xfff)
   10294: 64003805  bge $r0,$r5,56(0x38) # 102cc <bubble_sort+0x3c>
   10298: 004088a6  slli.w  $r6,$r5,0x2
   1029c: 00101886  add.w $r6,$r4,$r6
   102a0: 00150089  move  $r9,$r4
   102a4: 28800128  ld.w  $r8,$r9,0
   102a8: 28801127  ld.w  $r7,$r9,4(0x4)
   102ac: 64000ce8  bge $r7,$r8,12(0xc) # 102b8 <bubble_sort+0x28>
   102b0: 29800127  st.w  $r7,$r9,0
   102b4: 29801128  st.w  $r8,$r9,4(0x4)
   102b8: 02801129  addi.w  $r9,$r9,4(0x4)
   102bc: 5fffe8c9  bne $r6,$r9,-24(0x3ffe8) # 102a4 <bubble_sort+0x14>
   102c0: 02bffca5  addi.w  $r5,$r5,-1(0xfff)
   102c4: 02bff0c6  addi.w  $r6,$r6,-4(0xffc)
   102c8: 5fffd8a0  bne $r5,$r0,-40(0x3ffd8) # 102a0 <bubble_sort+0x10>
   102cc: 4c000020  jirl  $r0,$r1,0
```

和-O2的区别无法在此函数上体现

```
00010290 <bubble_sort>:
   10290: 02bffca5  addi.w  $r5,$r5,-1(0xfff)
   10294: 001500a8  move  $r8,$r5
   10298: 001120a9  sub.w $r9,$r5,$r8
   1029c: 64003925  bge $r9,$r5,56(0x38) # 102d4 <bubble_sort+0x44>
   102a0: 00150089  move  $r9,$r4
   102a4: 00150007  move  $r7,$r0
   102a8: 50002000  b 32(0x20) # 102c8 <bubble_sort+0x38>
   102ac: 28800126  ld.w  $r6,$r9,0
   102b0: 2880112a  ld.w  $r10,$r9,4(0x4)
   102b4: 64000d46  bge $r10,$r6,12(0xc) # 102c0 <bubble_sort+0x30>
   102b8: 2980012a  st.w  $r10,$r9,0
```

```
    102bc: 29801126  st.w  $r6,$r9,4(0x4)
    102c0: 028004e7  addi.w  $r7,$r7,1(0x1)
    102c4: 02801129  addi.w  $r9,$r9,4(0x4)
    102c8: 63ffe4e8  blt $r7,$r8,-28(0x3ffe4) # 102ac <bubble_sort+0x1c>
    102cc: 02bffd08  addi.w  $r8,$r8,-1(0xfff)
    102d0: 53ffcbff  b -56(0xfffffc8) # 10298 <bubble_sort+0x8>
    102d4: 4c000020  jirl  $r0,$r1,0
```

查资料以后认为此优化是：

> 对代码大小的优化，我们基本不用做更多的关心。 通常各种优化都会打乱程序的结构，让调试工作变得无从着手。并且会打乱执行顺序，依赖内存操作顺序的程序需要做相关处理才能确保程序的正确性。

---

4. ABI 中会包含对结构体中各元素的对齐和摆放方式的定义。

   a. 在你的机器上用 C 语言编写一段包含不同类型的结构体，并获得结构体总空间占用情况。

正好之前的操作系统课程布置了类似的作业，故直接拿来调整后使用

```
#include <stdio.h>

typedef struct {
    char c;
    short s;
    int i;
    long l;
    float f;
    double d;
    long double ld;
}struct_t1;

typedef struct {
    char c;
    short s;
    int i;
    long l;
    float f;
    double d;
}struct_t2;

typedef struct {
    char c;
    short s;
    int i;
    double d;
}struct_t3;

typedef struct {
    char c;
    int i;
    double d;
    short s;
}struct_t4;

typedef struct {
    int i;
    double d;
    char c;
```

```
    short s;
}struct_t5;

int main() {

    struct_t1 str1;
    struct_t2 str2;
    struct_t3 str3;
    struct_t4 str4;
    struct_t5 str5;

    printf("%lu\n", sizeof (str1));
    printf("%lu\n", sizeof (str2));
    printf("%lu\n", sizeof (str3));
    printf("%lu\n", sizeof (str4));
    printf("%lu\n", sizeof (str5));

    return 0;
}
```

```
/Users/zhoupengyu/Documents/homeworks/2021OS_Assignments/size_of_int/cmake-build-debug/size_of_int
48
32
16
24
24
```

b. 调整结构体元素顺序，推测并分析结构体对齐的方式。

结构体以其中包含的最长元素为单位对齐。

---

5. 写一个可以打印出输入字符的程序（汇编，嵌入式或者直接系统调用），单步调试并观察变化，对照平台ABI给出解释

```
.section .rodata
.lcomm  buff, 1
.text
.global _start
_start:
        #number:3 sys_read(0,buff,1)
  movl $3, %eax
        movl $0, %ebx    # 1st param
        movl $buff, %ecx # 2st param
        movl $1, %edx    # 3st param
        int $0x80

        #number:4 sys_write(1)
  movl $4, %eax
        movl $1, %ebx  # 1st param
        int $0x80

  # exit(0)
  movl $1, %eax
  movl $0, %ebx  # 1st param
  int $0x80
```

其ABI规则为：`%eax`寄存器传递系统调用号，`%ebx`，`%ecx`等传递参数，可以注意到，在调用sys_read时，向`%eax`寄存器中存的数是3，即其系统调用号为，而后存入buff的地址以及大小

于`%ecx`，`%edx`寄存器中，之后即可执行系统调用。