实验二

实验报告

计算机科学与技术系

191220008 陈南曈

一、实验目的

1、了解并学习计算机的数据表示方式，了解并学习计算机的算术运算方式，理解不同 数据类型的运算属性。

2、了解并学习 gdb 的使用方法，并运用其进行内存、寄存器检查。

二、实验内容

1、在 64 位计算机中运行一个 C 语言程序，在该程序中出现了以下变量的初值，请在 表格中填写它们对应的机器数（用十六进制表示）。

变量	x	y	z	c
机器数	0xffff80000	0x020a	0x0000fffa	0x40
变量	a	b	u	v
机器数	0xbf8ccccd	0x40250000 00000000	0x4e932c06	0x41d26580 b4800000

```
++++++++++Machine value++++++++++++++++
x = 0xffff8000
y = 0x20a
z = 0xfffa
c = 0x40
a = 0xbf8ccccd
b = 0x4025000000000000
u = 0x4e932c06
v = 0x41d26580b4800000
++++++++++Real value++++++++++++++++++
x = -32768
y = 522
z = 65530
c = @
a = -1.100000
b = 10.500000
u = 1234567936.000000
v = 1234567890.000000
```

```
(gdb) x/1xw &x
0x7fffffffdd30: 0xffff8000
(gdb) x/1xh &y
0x7fffffffdd26: 0x020a
(gdb) x/1xw &z
0x7fffffffdd34: 0x0000fffa
(gdb) x/1xb &c
0x7fffffffdd25: 0x40
(gdb) x/1xw &a
0x7fffffffdd28: 0xbf8ccccd
(gdb) x/1xg &b
0x7fffffffdd38: 0x4025000000000000
(gdb) x/1xw &u
0x7fffffffdd2c: 0x4e932c06
(gdb) x/1xg &v
0x7fffffffdd40: 0x41d26580b4800000
```

2、使用命令 gcc –ggdb swap.c –o swap 编译下面的 swap.c 代码，完成后面的实验

1) 使用 gdb 命令查看程序变量的取值，填写下面两个表格:

a 的存放地址（&a）	b 的存放地址（&b）	x 的存放地址（&x）	y 的存放地址（&y）
0x7fffffffdd30	0x7fffffffdd34	0x7fffffffdd18	0x7fffffffdd10

```
(gdb) p &a
$1 = (int *) 0x7fffffffdd30
(gdb) p &b
$2 = (int *) 0x7fffffffdd34
```

```
(gdb) p &x
$1 = (int **) 0x7fffffffdd18
(gdb) p &y
$2 = (int **) 0x7fffffffdd10
```

执行步数	x 的值（机器值，用十六进制）	y 的值（机器值，用十六进制）	*x 的值（程序中的真值，用十进制）	*y 的值（程序中的真值，用十进制）
第一步前	0x7fffffffdd30	0x7fffffffdd34	01	02
第一步后	0x7fffffffdd30	0x7fffffffdd34	01	03
第二步后	0x7fffffffdd30	0x7fffffffdd34	02	03
第三步后	0x7fffffffdd30	0x7fffffffdd34	02	01

第一步前:

```
(gdb) p/x x
$3 = 0x7fffffffdd30
(gdb) p/x y
$4 = 0x7fffffffdd34
(gdb) p/o *x
$5 = 01
(gdb) p/o *y
$6 = 02
```

第一步后:

```
(gdb) p/x x
$7 = 0x7fffffffdd30
(gdb) p/x y
$8 = 0x7fffffffdd34
(gdb) p/o *x
$9 = 01
(gdb) p/o *y
$10 = 03
```

第二步后：

```
(gdb) p/x x
$11 = 0x7fffffffdd30
(gdb) p/x y
$12 = 0x7fffffffdd34
(gdb) p/o *x
$13 = 02
(gdb) p/o *y
$14 = 03
```

第三步后：

```
(gdb) p/x x
$15 = 0x7fffffffdd30
(gdb) p/x y
$16 = 0x7fffffffdd34
(gdb) p/o *x
$17 = 02
(gdb) p/o *y
$18 = 01
```

2) 运行下面的 reverse.c，并说明输出这种结果的原因，修改代码以得到正确的逆序数组：

结果：

```
7 6 5 0 3 2 1
```

原因：当数组长度为奇数时，存在 left 和 right 相等的情况。在 left 和 right 相等时，xor_swap 函数传入的两个参数将对应同一个数组元素。在执行完 xor_swap 函数时，*x 和*y 是同一个元素，最后一步的*y 将该数组元素将被修改为 0。故修改时，应该避免 left 和 right 相等时执行 xor_swap 函数。

修改后结果：

```
7 6 5 4 3 2 1
```

3、编译并运行下面的程序，使用 gdb 指令查看变量的取值，解释语句输出为 False 的原因并填写在表格中

	输出 True/False	原因
语句一	True	
语句二	False	float 精度太低，尾数的最后四舍五入后发生改变，再转回 int 后与原来不相同
语句三	False	float 类型的精度为 6-7 位有效数字，比 double 少。p1 与 p2 在 float 精度范围内的数字相同，故 p1 与 p2 相等
语句四	True	
语句五	False	f 太大，d+f 时 d 被忽略，故最后等于 0

```
++++++++++True or False++++++++++++++++
x==(int)xd True
x==(int)xf False
p1!=p2 False
result1==d True
result2==d False
```

4、观察下面 data_rep.c 程序的运行:

1) 使用命令 gdbtui data_rep 进入 gdb 的 TUI 调试模式, 之后分别输入命 令: layout asm 和 layout regs, 再输入命令 start 启动程序, 然后使用 si 命令进行单 步运行。请在单步运行过程中完成下面的表格

	机器数（十六进制）	真值（十进制）		机器数（十六进制）	真值（十进制）
x	0x66	102	x	0x39	57
~x	0x99	153	!x	0x00	0
x & y	0x20	32	x && y	0x1	1
x | y	0x7f	127	x || y	0x1	1

	机器数（十六进制）	真值（十进制）	OF	SF	CF	AF
x1	0x7fffffff	2147483647	0	0	0	0
y1	0x1	1	0	0	0	0
sum_x1_y1	0x80000000	-2147483648	1 (int 上界溢出)	1（结果最高位为 1）	0	1（发生进位）
diff_x1_y1	0x7ffffffe	2147483646	0	0	0	0
diff_y1_x1	0x80000002	-2147483646	0	1（结果最高位为 1）	1（最高位产生借位）	1（发生进位）
x2	0x7fffffff	2147483647	0	0	0	0
y2	0x1	1	0	0	0	0
sum_x2_y2	0x80000000	2147483648	1（最高位进位）	1（结果最高位为 1）	0	1（发生进位）
diff_x2_y2	0x7ffffffe	2147483646	0	0	0	0
diff_y2_x2	0x80000002	2147483650	0	1（结果最高位为 1）	1（最高位产生借位）	1（发生进位）