1.

变量	x	y	z	c
机器数	0xffff8000	0x020a	0x0000fffa	0x40
变量	a	b	u	v
机器数	0xbf8ccccd	0x4025000000000000	0x4e932c06	0x41d26580b4800000

代码输出的结果不带前导0，而gdb显示的结果带有前导0，该表以gdb显示的结果为准.

2.

1)

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)
0xbffff074	0xbffff078	0xbffff068	0xbffff06c

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	0xbffff074	0xbffff078	1	2
第一步后	0xbffff074	0xbffff078	1	3
第二步后	0xbffff074	0xbffff078	2	3
第三步后	0xbffff074	0xbffff078	2	1

2)

输出结果:

```
noob@noob-virtual-machine:~/workspace/lab02$ ./reverse
7 6 5 0 3 2 1
```

原因:

在函数 reverse_array 的最后一次循环中，有 &a[left] = &a[right]. 这导致在调用 xor 交换函数时，该函数第一步操作会导致 a[left]和 a[right]的值同时被修改为 0，且后续操作无法弥补这个错误，于是产生了如图所示的输出情况.

3.

	输出 True/False	原因
语句一	True	double 的 frac 有 52-bits, 大于 int 类型的数据长度 32-bits, 不会丢失信息.
语句二	False	float 的 frac 只有 23-bits, 而 x 的值为 INT_MAX 需要完整的 32-bits 才能表示, 所以在转换时会丢失信息, 导致结果错误.
语句三	False	float 类型有效位为 6 - 7 位, 无法精确表示 3.141592653 和 3.141592654.
语句四	True	显然
语句五	False	由于 d 很小 f 很大, 导致(f+d)被舍入后为 f, f – f = 0 不为 d, 所以判断错误.

4.

1)

	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	y	0x39	57
~x	0x99	-103	!x	0x00	0
x & y	0x20	32	x && y	0x01	1
x | y	0x7f	127	x || y	0x01	1

2)

	机器数 (十六进制)	真值 (十进制)	OF	SF	CF	AF
x1	0x7fffffff	2147483647	0	0	0	0
y1	0x00000001	1	0	0	0	0
sum_x1_y1	0x80000000	-2147483648	1(运算结果超出有符号整数能表示的范围)	1(运算结果的最高位为“1”)	0	1(运算时半字节产生进位)
diff_x1_y1	0x7ffffffe	2147483646	0	0	0	0
diff_y1_x1	0x80000002	-2147483646	0	1(运算结果的最高位为“1”)	1(有借位)	1(运算时半字节产生借位)
x2	0x7fffffff	2147483647	0	0	0	0
y2	0x00000001	1	0	0	0	0
sum_x2_y2	0x80000000	2147483648	1(运算结果超出有符号整数能表示的范围)	1(运算结果的最高位为“1”)	0	1(运算时半字节产生进位)
diff_x2_y2	0x7ffffffe	2147483646	0	0	0	0
diff_y2_x2	0x80000002	2147483650	0	1(运算结果的最高位为“1”)	1(有借位)	1(运算时半字节产生借位)