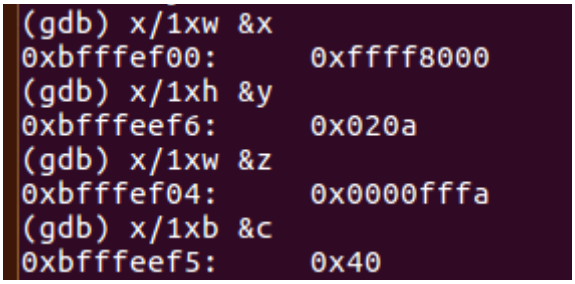
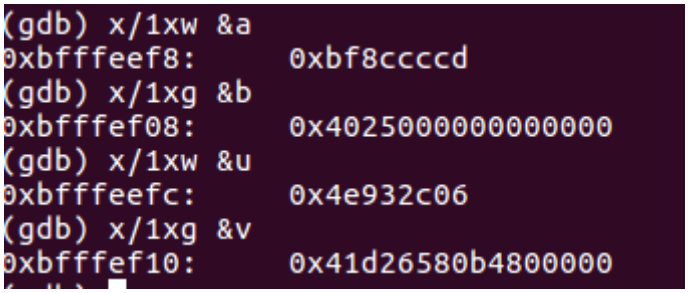
1.

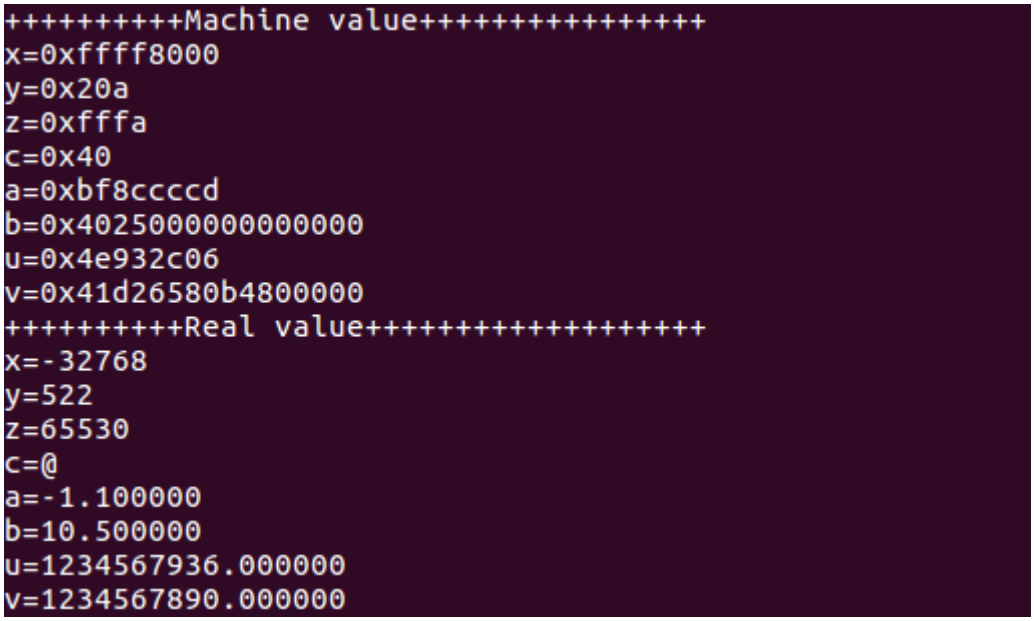
variable	x	y	z	c
Machine number	0xffff8000	0x020a	0x0000fffa	0x40
variable	a	b	u	v
Machine number	0xbf8ccccd	0x4025000000000000	0x4e932c06	0x41d26580b4800000

The result of GDB:





The result of running C program:



While the result of both is the same .

2.

I.

The address of a(&a)	The address of b(&b)	The address of x(&x)	The address of y(&y)
0xbfffef24	0xbfffef28	0xbfffef18	0xbfffef1c

Steps	x	y	*x	*y
Before 1st step	0xbfffef24	0xbfffef28	1	2
After 1st step	0xbfffef24	0xbfffef28	1	3
After 2nd step	0xbfffef24	0xbfffef28	2	3
After 3rd step	0xbfffef24	0xbfffef28	2	1

ii.

```
hyc@161220049:~/workspace/lab02/161220049$ gcc -o reverse reverse.c
hyc@161220049:~/workspace/lab02/161220049$ ./reverse
7650321
hyc@161220049:~/workspace/lab02/161220049$
```

Reason of error: Becaude the number ‘4’ is in the middle, when we run xor_swap(&a[3],&a[3]), a[3] is always the same as a[3]. So

when head = tail = (len-1)/2, **x and y share the same memory,** which makes **x**=y=0 after doing “*x = *x ^ *y”

The changed version:

```c
#include <stdio.h>
void xor_swap(int *x,int *y){
    *y=*x ^ *y;
    *x=*x ^ *y;
    *y=*x ^ *y;
}

void reverse_array(int a[],int len){
    int left,right=len-1;
    while (left<right)
        {
            xor_swap(&a[left],&a[right]);
            left++;
            right--;
        }
}

int main(){
    int a[]={1,2,3,4,5,6,7};
    reverse_array(a,7);
    int i;
    for (i=0;i<7;i++)
        printf("%d",a[i]);
    printf("\n");
}
```

3.

	Output: True/False	Reason
Sentence 1	True	Int is 4 bytes,double is 8 bytes, when changing from double to int, it only cuts the partion part, and has nothing to do with integer part.
Sentence 2	False	Int is 4 bytes,float is 4 bytes,too.(int)xf will be -2147483648 instead of 2147483647.
Sentence 3	False	After cutting some bits, p1 and p2 will both become 3.141593 and they share the same value.
Sentence 4	True	Because of precision problem, f-f will become 0, so result1==1.0, and it is the same as d.
Sentence 5	False	Because of precision problem, (d+f)=f, so result2 will become 0 instead of 1.0.

4.

1)

	Machine number(hex)	Value(dec)		Machine number(hex)	Value(dec)	
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	
	Machine number(hex)	Value(dec)	OF	SF	CF	AF
x1	0xb0	-80	0	0	0	0
y1	0x8c	-116	0	0	0	0
x1+y1	0x3c	-196	0	0	0	0
x1-y1	0x24	36	0	0	0	1
y1-x1	0xdc	-36	0	1	1	0
x2	0xb0	176	0	1	1	0
y2	0x8c	140	0	1	1	0
x2+y2	0x3c	316	0	0	0	0
x2-y2	0x24	36	0	0	0	1
y2-x2	0xdc	-36	0	1	1	0

The reasons of the change of eflags:

(1) SF is set as MSB. Because y1-x1, y2-x2 is negative and x2+y2 is positive, and SF changes only when performing operations.
(2) CF is a flag indicating carrying-bit or borrowing-bit. When we do y1-x1 or y2-x2, we should borrow bits, so the CF changes.
(3) AF is auxiliary carrying flag. When we do x1-y1 or x2-y2,a carry happens to bit 3rd