Write a subprogram `sum_even` that finds the sum of even numbers in an array. This subprogram gets as argument IN base address of an array and array size and returns the sum as argument OUT. Arguments IN and OUT should be passes in appropriate registers.

Write a main code that allocate a static array named: `array_p` and initializes it to: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} and static variable named: `size_p` initializes it to 10. Thereafter, subprogram passes base address of the static array and value of array size into subprogram `sum_even` and prints the returned sum of even numbers.

Given that **`$t1 = 0xF7AB1BFF`**, **`$t2 = 0xA976FBEE`** what value is stored in `$t0` after the bit operation is completed.

```
addi $t0, $t1, 0xE3A1
```

```
or $t0, $t1, $t2
```

```
xor  $t0, $t1, $t2
```

```
xori $t0, $t2, 0x93CC
```

```
andi $t0, $t2, 0x93BB
```

CS 315 Lab 6

Write a subprogram `sum_even` that finds the sum of even numbers in an array. This subprogram gets as argument IN base address of an array and array size and returns the sum as argument OUT. Arguments IN and OUT should be passes in appropriate registers.

```
      .data
      .text
sum_even:
      move $t0, $a0    # base address
      move $t1, $a1    # array size
      li $t2, 0        # initialize sum to zero
      li $t3, 2        # constant value 2

sum_even_loop:
      blez $t1, sum_even_end          # if count <= 0 branch to end

      lw $t4, 0($t0)                  # fetch array value from memory
      rem $t5, $t4, $t3               # divide array value by 2
      bgtz $t5, sum_even_loop_skip    # if remainder > 0 skip (or odd)
      add $t2, $t2, $t4               # add number to the sum

sum_even_loop_skip:
      addi $t0, $t0, 4                # increment array base address
      addi $t1, $t1, -1               # decrement array size

      b sum_even_loop                 # branch unconditionally back to loop

sum_even_loop_end:
      move $v0, $t2                   # return sum
      jr $ra                          # jump back to main
```

Write a main code that allocate a static array named: `array_p` and initializes it to: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} and static variable named: `size_p` initializes it to 10. Thereafter, subprogram passes base address of the static array and value of array size into subprogram `sum_even` and prints the returned sum of even numbers.

```
      .data
array_p:   .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
size_p:    .word 10
      .text
main:
      la $a0, array_p  # load base address of static array into $a0

      la $t9, size_p   # load address of static variable into $t9
      lw $a1, 0($t9)   # $a1 <-- memory[$t9 + 0]

      jal sum_even     # call subprogram sum_even
```

```
    move $t0, $v0    # copy sum into $t0

    li v0, 1         # system call 1 to print integer
    move $a0, $t0    # copy sum into $a0 to be printed
    Syscall

    li v0, 10        # halt
    syscall
```

Given that `$t1 = 0xF7AB1BFF`, `$t2 = 0xA976FBEE` what value is stored in `$t0` after the bit operation is completed.

```
1111 0111 1010 1011 0001 1011 1111 1111 <-- $t1
1010 1001 0111 0110 1111 1011 1110 1110 <-- $t2

addi $t0, $t1, 0xE3A1

1111 1111 1111 111         111 1111 111
1111 0111 1010 1011 0001 1011 1111 1111 <-- $t1
1111 1111 1111 1111 1110 0011 1010 0001 <-- 0xE3A1
---------------------------------------
1111 0111 1010 1010 1111 1111 1010 0000 <-- 0xF7AAFFA0


or $t0, $t1, $t2
1111 0111 1010 1011 0001 1011 1111 1111 <-- $t1
1010 1001 0111 0110 1111 1011 1110 1110 <-- $t2
---------------------------------------
1111 1111 1111 1111 1111 1011 1111 1111 <-- 0xFFFFFBFF

xor  $t0, $t1, $t2
1111 0111 1010 1011 0001 1011 1111 1111 <-- $t1
1010 1001 0111 0110 1111 1011 1110 1110 <-- $t2
---------------------------------------
0101 1110 1101 1101 1110 0000 0001 0001 <-- 0x5EDDE011

xori $t0, $t2, 0x93CC

1010 1001 0111 0110 1111 1011 1110 1110 <-- $t2
0000 0000 0000 0000 1001 0011 1100 1100 <-- 0x93CC
---------------------------------------
1010 1001 0111 0110 0110 1000 0010 0010 <-- 0xA9766822

andi $t0, $t2, 0x93BB
1010 1001 0111 0110 1111 1011 1110 1110 <-- $t2
0000 0000 0000 0000 1001 0011 1011 1011 <-- 0x93BB
---------------------------------------
0000 0000 0000 0000 1001 0011 1010 1010 <-- 0x93AA
```