Question 1

What is the machine code for the following instructions:

```
ADD x16, x0, x0
ADDI x2, x1, -16
LHU, x4, x3, -4
JALR x0, 8(x1)
```

Response

The machine code for the following instructions are:

```
00000000 00000 00000 000 10000 0110011
111111110000 00001 000 00010 0010011
111111111100 00011 101 00100 00000011
000000001000 00001 000 00000 1100111
```

Question 2

What is the machine code for the following JAL instruction:

```
X100 jal x0, Loop
...
X200 Loop:
    ...
```

Response

The machine code for the following JAL instruction is:

```
0 00000000000 0 1100100 00001 1101111
```

Question 3

Translate the following machine codes into RISC-V assembly language (numbers are in hex).

`fe1ff06f`
`0000c133`

Response

The instruction `fe1ff06f` translates to:
`1111 1110 0001 1111 1111 0000 0110 1111` which, when rearranged, yields:
`1 11111100001 1 1111111 00000 1101111` which is a J-type, so we get:
`1 1111111 1 11111100001 00000 1101111` (`imm[20|10:1|11|19:12]`) or:
`jal -31(x0)`

The instruction `0000c133` translates to:
`0000 0000 0000 0000 1100 0001 0011 0011` which, when rearranged, yields:
`0000000 00000 00001 100 00010 0110011` which is:

`xor x2 x0` in RISC-V assembly.

Question 4

Write a RISC-V assembly language program for determining if a number is even or not.

Use the following information in writing your assembly code.

1. The function starts at memory location `0x400`. Each instruction is 32 bits, thus the second instruction should start at `0x404`, and so on. Use this information to correctly compute the offset for jump and branch instructions (you are not allowed to use labels).
2. The input is passed (stored) in register `a0`.
3. The return value, c, should be stored in `a0`.
4. The return address is tored in `ra`.
5. You are free to use saved and temporary registers (don't forget to save values if you are using saved registers).
6. You are allowed to use Pseudoinstructions (e.g. `ret`, `call`, etc.)

```
addi t0, a0, 0
addi t0, t0, -2
blt zero, t0, -4
sub a0, a0, a0
blt zero, t0, 8
addi t0, t0, 2
xor a0, a0, t0
xori a0, a0, 1
```