Address	Code	Code in Binary							
		31 28	27 24	23 20	19 16	15 12	11 8	7 4	3 0
0x00400000	0x0114a403	0000	0001	0001	0100	1010	0100	0000	0011
0x00400004	0x00842403	0000	0000	0100	0100	0010	0100	0000	0011
0x00400008	0x01398e63	0000	0001	0011	1001	1000	1110	0110	0011
0x0040000c	0x007302b3	0000	0000	0111	0011	0000	0010	1011	0011
0x00400010	0x0180006f	0000	0001	1000	0000	0000	0000	0110	1111
0x00400014	0x01399863	0000	0001	0011	1001	1001	1000	0110	0011
0x00400018	0x01ee8e33	0000	0001	1110	1110	1000	1110	0011	0011
0x0040001c	0x00231293	0000	0000	0010	0011	0001	0010	1001	0011
0x00400020	0x000009b3	0000	0000	0000	0000	0000	1001	1011	0011
0x00400024	0xff4980e3	1111	1111	0100	1001	1000	0000	1110	0011
0x00400028	0x008000ef	0000	0000	1000	0000	0000	0000	1110	1111
0x0040002c	0x00008067	0000	0000	0000	0000	1000	0000	0110	0111

Table 1: A segment of RISC-V assembly program showing the address where each instruction is stored in memory, the hexadecimal and the binary representation of the instruction.

Question 4 (12 points):

Table 1 shows both the hexadecimal and binary instruction representations for several instructions next to the memory address where each instruction is stored. The hexadecimal representation of all the addresses start with the pattern 0x004000 thus in the table below we have printed these digits so that you only need to write the last two digits for each address. In the table below identify the address that contains the instruction described. (Hint: You do not need to completely decode each instruction to determine the binary representation of the instruction that corresponds to the description. In some cases you only need to check the opcode and a few other fields)

Description	Address
An instruction that multiplies the value in a register by the constant 4.	0x0040001c
A load word where the address register is the same as the destination register	0x00400004
An unconditional jump	0x00400010
A function call	0x00400028
A load word with an odd value for offset	0x00400000
A backward branch	0x00400024
A return statement	0x0040002c
An add instruction that sets the value of a register to zero	0x00400020

A time-saving strategy during the exam is to realize that there is no need to fully decode the instruction binaries in order to answer the question.

- **An instruction that multiplies the value in a register by the constant 4** We are looking for a `slli, xx, xx, 2` instruction which has an I format, the opcode is `0010011` and the function code is `001`. The function code is in bits `14-12`. The address is `0x0040001C`.
- **A load word where the address register is the same as the destination register** Opcode = `0000011`. Format I. Function Code in bits `14-12` is `010` Bits `19-15` and `11-7` must be the same. Address `0x00400004`.
- **An unconditional jump**: It must be a `jal` instruction with the destination register equal zero. The opcode for is `1101111`. There are two binaries with this opcode: `0x0180006f` and `0x008000ef`. Bits `11-7` are the destination register and must be zero. Address is `0x00400014`.

  0x01399863 0x008000ef
- **A function call**: Similar reasoning but destination register cannot be zero. Answer: address is `0x00400028`.
- **A load word with an odd value for offset**: Opcode is `0000011`. Bit 20 must be 1. Address is `0x00400000`.
- **A backward branch**: Bit 31 must be 1. Address `0x00400024`.
- **A return statement**: jalr is an I format instruction. Opcode must be `1100111` and bits 19-15 must be `00001` to indicate `rs1 = ra =x1`. Address `0x0040002c`
- **An `add` instruction that sets the value of a register to zero**: Opcode for `add` is `0110011`, format is R, function code in bits `14-12` is `000` and both source registers in bits `24-20` and `19-15` must be 0. Address `00400020`

Below is the code segment shown in Table 1.

```
    lw   s0, 17(s0)      # load word with an odd value for offset
target2:
    lw   s0, 8(s0)       # load word where the address register is the same as the destination
    beq  s3, s3, target1 # a conditional branch that is always taken
    add  t0, t1, t2
    jal  zero, target3   # an unconditional jump
    bne  s3, s3, target1 # a conditional branch that is never taken
    add  t3, t4, t5
    slli t0, t1, 2       # an instruction that multiplies the value in a register by 4
    add  s3, zero, zero  # An instruction that sets the value of a register to zero
target1:
    beq  s3, s4, target2 # a backward branch
target3:
    jal  ra, bar         # a function call
    ret                  # a return statement
```