The time that it takes to decode an instruction and read the value of a register to determine the return address when executing a return instruction in a modern processor is too long — it would cause the processor to miss several cycles in which it cannot fetch instructions. Thus, all modern processors implement a Return Address Stack (RAS) in hardware. For instance the new AMD ZEN 5 processors feature a RAS with 52 entries. Whenever a function-call instruction is executed, its return address is pushed, by the hardware, into RAS. Whenever a return instruction is decoded, the hardware pops the top of the stack and starts fetching instructions from that address.

To simulate a RAS in software, the instructions to push and pop into the simulated RAS would have to be inserted in an existing binary file for a RISC-V program. However, inserting instructions into an existing program is too complex a task for a midterm question — CMPUT 229 students will perform this task in their lab #6 this term.

In this exam, we will write two functions that support the implementation of a RAS simulator. These functions require binary vectors with arbitrary lengths. Implementing a library to create and manipulate such vectors is also too complex for the time of this exam. Thus, assume that such a library already exists.

You are asked to write two functions: `CallReturn` and `RAS`. `CallReturn` determines if an instruction is a function call, a return statement, or something else.

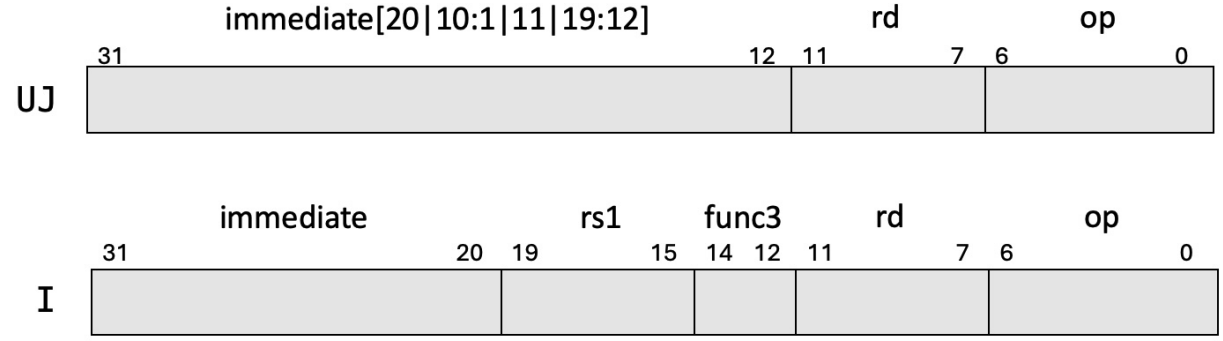

Figure 1: UJ and I instruction formats.

In RISC-V, a function call is performed with the instruction `jal ra, LABEL`. To be a function call, the destination register of the `jal` instruction must be `ra = x1`. The opcode for `jal` is `1101111`. The `jal` instruction uses the `UJ` format shown in the top of Figure 1.

A return statement in RISC-V is executed by the instruction `jalr rd, ra, immed`, which uses the I format, shown in the bottom of Figure 1. Any register can be used for `rd` and any value can be the immediate (an exception is generated if the computed address is misaligned but we will not check for such condition in this question). A `jalr` instruction is deemed to be a return statement for RAS purpose if the source (`rs1`) register is `ra`.

The assembly code that you write for both parts of this question must follow all the register saving/restoring conventions for RISC-V.

Question 5 (20 points): Write RISC-V assembly for the function `CallReturn` with the following specification:

- Arguments:
  - `a0`: address of a RISC-V instruction
- Return Value:
  - `a0 = 1`: if the instruction is a function call instruction
  - `a0= -1`: if the instruction is a return statement
  - `a0= 0`: if it is any other type of instruction

```
322 CallReturn:
323             lw   t0, 0(a0)         # t0 <- binary of the instruction
324             li   a0, 1             # a0 <- register number for ra
325             slli t1, t0, 20        # t1 <- Rs and Opcode in upper bits
326             lui  t2, 0x0EF00       # t2 <- Rs=00001 and jal opcode 1101111 in upper bits
327             bne  t1, t2 NotCal
328             jalr x0, ra, 0         # return a0 = 1
329 NotCal:     slli t3, t0, 25
330             lui  t4, 0xCE000       # t4 <- jalr opcode in upper bits 1100 1110
331             beq  t3, t4 isJalr
332             mv   a0, zero
333             jalr x0, ra, 0         # return a0 = 0
334 isJalr:     slli t6, t0, 12
335             srli t6, t6, 27        # t6 <- five bits specifying ra
336             bne  t6, a0, NotReturn # if t6 != 1 it is not a return statement
337             li   a0, -1
338             jalr x0, ra, 0         # return a0 = -1
339 NotReturn: mv   a0, zero
340             jalr x0, ra, 0         # return a0 = 0
```

Figure 2: A solution for `CallReturn`.