1. What instruction would you use to subtract a constant, say 8, from a register?

   (a) `add`

   (b) `addi`

   (c) `sub`

   (d) `subi`

   Answer: (b)

2. Which MIPS instruction would you use (ideally) to multiply the value of a register by 4?

   (a) `mul`

   (b) `muli`

   (c) `add`

   (d) `addi`

   (e) `sll`

   (f) `srl`

   Answer: (e)

3. What MIPS instruction can be used to divide the value of a register by 4 to get the quotient?

   (a) `muli`

   (b) `divi`

   (c) `sll`

   (d) `srl`

   Answer: (d)

4. Which MIPS instruction can be used to divide the value of a register by 8 and get the remainder?

   (a) `divi`

   (b) `sll`

(c) `and`

(d) `andi`

Answer: (d)

5. What instruction or sequence of instructions would you use to compare two registers for equality, and branch based on the comparison?

   (a) `bez`

   (b) `bnz`

   (c) `beq`

   (d) `slt`

   Answer: (c)

6. In the example given in slides, is it possible to support the function call (not the return), using the MIPS instructions we have seen so far?

   (a) Yes

   (b) No

   Answer: (a)

7. What instruction would you use to subtract a constant, say 8, from a register?

   (a) `add`

   (b) `addi`

   (c) `sub`

   (d) `subi`

   Answer: (b)

8. Why is `$ra` set to PC + 4 in the `jal` instruction?

   Answer: `$ra` is set to PC + 4 in the `jal` instruction because PC contains the address of the current instruction, and the next instruction, which follows the `jal`, will be at PC + 4. Therefore, storing PC + 4 in `$ra` provides the correct return address for when the function or procedure called by `jal` completes and control needs to be returned to the point just after the `jal` instruction.

9. We used the mechanism of using `jal` and `jr` for function call and return respectively. Will this mechanism alone be sufficient to support nested function calls?

   (a) `Yes`

   (b) `No`

   Answer: (b)

10. Will the use of two return address registers solve the problem of `$ra` getting overwritten during a nested function call? That is, will it solve the problem for all possible nested function calls?

    (a) Yes

    (b) No

    Answer: (b)

11. Where can we store the return addresses in the case of arbitrary nesting of function calls? Think in terms of a data structure.

    Answer: Call Stack

12. Will the second option of saving `$ra` at the beginning of f() and restoring it just before returning from f(), work correctly?

    (a) Yes

    (b) No

    Answer: (a)

13. Which of the two options is more efficient?

    (a) First option: saving and restoring for each nested call

    (b) Second option: saving and restoring (atmost) once in f()

    Answer: (b)

14. What is the address of the top-most word in MIPS memory? (Give your answer in small-case hex digits, without the preceding 0x).

    Answer: `fffffffc`