2 Compiler Construction (jdy22)

A library for a Slang-like language supports explicit lazy evaluation via a type `lazy` and functions `delay` and `force`:

```
type 'a cell =
| Val of 'a
| Exn of exn
| Lazy of (unit -> 'a)
type 'a lazy =
    ('a cell) ref
```

```
let delay f = ref (Lazy f)
let force r = match !r with
  | Val v -> v
  | Exn e -> raise e
  | Lazy f -> try (let v = f () in r := Val v; v)
              with e -> r := Exn e; raise e
```

You decide to incorporate similar support for lazy evaluation into the language, adding built-in constructs `delay e` and `force e`, where `e` is an expression.

(*a*) Outline the benefits and drawbacks of implementing laziness in the compiler rather than in a library. [4 marks]

(*b*) Give new Jargon VM instructions that can implement `delay` and `force` and describe their behaviour. [6 marks]

(*c*) Give the translation of the `delay` and `force` constructs into your extended instruction set. [6 marks]

(*d*) You now consider adding an optimization that evaluates the argument of `delay` eagerly rather than creating a delayed computation.

(*i*) Give an expression `e` for which the transformation is valid (that is, behaviour-preserving) and always an optimization. [1 mark]

(*ii*) Give an expression `e` for which the transformation is valid and only sometimes an optimization. [1 mark]

(*iii*) Give an expression `e` for which the compiler cannot ascertain whether the transformation is valid. [1 mark]

(*iv*) Give an expression `e` for which the transformation is not valid. [1 mark]