COMP.2030 Homework 2

Consider an array, X, containing only nonnegative integers. The last value in X is set to `-1` to mark the end of the array.

For simplicity, suppose that the search key, `key`, is global. A linear search program can be written as on the left. The same logic is written with '`goto`'s on the right, called pseudo-C or C-- code.

```
void main() {
    int i = 0;
    int found = 0;
    while (!found && X[i] != -1) {
        if (X[i] == key) {
            found = 1;
        } else {
            i++;
        }
    }
    if (found) {
        printf("%d", i-1);
    } else {
        printf("-1");
    }
}
```

```
void main() {
    int i = 0;
    int found = 0;

  loop:
    if (found != 0) goto done;
    if (X[i] == -1) goto done;

    if (X[i] != key) goto else;
        found = 1;
        goto loop;
  else:
        i++;
        goto loop;

  done:
    if (found == 0) goto else2;
        printf("%d", i-1);
        goto exit;

  else2:
        printf("-1");

  exit:
    return;
}
```

Rewrite this C-- code into a "pseudo-assembly" version with registers instead of variables, simple expressions instead of complex expressions, explicit memory accesses, and only flag-checking conditionals.

- Global variable names, like X and `key`, as well as local variable names, like '`i`' and '`found`' and '`key`', should be replaced with a register name, `reg0`, `reg1`, `reg2`, etc.
- Expressions should only be to the right-hand side of an assignment, and should only be
    1. A single constant, like `0`,
    2. A single register, like `reg1`,
    3. A single binary operation, like + or ==, applied to only constants or registers

  Shorthand operators like '`i++`' should be replaced by '`i = i+1`'.
- For a memory access, use the standard C notation of pointer access, writing '`reg1 = *a`' to load the value stored at address `a` into register `reg1`, and writing '`*a = reg1`' to store the value in register `reg1` at address `a`. You may also include a numeric offset with memory access using the C notation for array access, writing '`reg = a[i]`' to load the

value stored at address `a+i` into register `reg1`, and writing '`a[i] = reg1`' to store the value in register `reg1` at address `a+i`.

- For a Boolean variable, use `0` for false and `1` for true.
- A conditional '`if (test) goto label`' statement in pseudo-C should be rewritten into two instructions
    1. A comparison operation '`comp(x,y)`'which will set global comparison flags (EQ, NEQ, LT, GT) remembering how the value of `x` compares to `y`, to be used next by
    2. An '`if (flag) goto label`' which can only test one global flag (such as EQ, NEQ, LT, GT) to decide whether or not to `goto` the given `label`.

  For example, `if (x < y) goto skip` would be rewritten as the two instructions:

```
comp(x, y);
if (LT) goto skip;
```

  and '`if (found != 0) goto done`' would be rewritten as the two instructions:

```
comp(found, 0);
if (NEQ) goto done;
```

- Calls to `printf` follow the same rules as binary operations. Its first argument should be any constant string or register, and its following 0 or more arguments should be a combination of constant values or register names.

What to submit:

1. Pseudo-assembly translated version of the given C-- code, rewritten following the rules described above, in a plain text, docx, or pdf file.
2. A document (either separate or in the same file) written in your own words explaining exactly how you approached the translation, how you handled each logical unit of the original program, and an argument of *why* your translation should do the same thing as the original version.