ECOAR lab – MIPS project winter 2018/2019

The black and white image is encoded using one bit per pixel. The oldest bit in the byte corresponds to the leftmost pixel in the image. The image line is aligned in the file to the nearest multiple of 4 bytes. In the black and white image we have following dependencies:
Number of bytes containing image pixels = (image width + 7)/8
Number of bytes in line (in file) = (((image width + 7)/8 + 3)/4) * 4

Information about image memory and important drawing parameters are stored in the imgInfo structure:

```
struct {
      int w, h;   // width and height of image
      unsigned char* pImg; // pointer to the image buffer
      // any additional values you think important (file size, for example)
} imgInfo;
```

Additionally the following structure will be useful:

```
struct {
      int x, y;   // horizontal and vertical coordinates of a point
} Point;
```

Implement the following function:

```
Point* FindPattern(imgInfo* pImg, int pSize,
                        int* ptrn, Point* pResult);
```

where

pSize size of the pattern in pixels (16 higher bits – horizontal size sx, 16 lower bits – vertical size sy); you can assume $5 \le sx \le 8$ and $5 \le sy \le 8$

ptrn pointer to the table containing subsequent lines of the pattern (on lowest bits)

pResult pointer to the table of Point structures, in which result will be stored (we assume that the table has sufficient size, i.e. we are guaranteed that function will not write outside the allocated memory). This pointer will be returned in the $v0 register (the $v1 register will contain the number of instances of the template in the image - in the example code in C it is an additional parameter of type int *)

The sample implementation of the FindPattern function in the graph_io.c file is far from perfect. Above all, the idea of looking for a pattern by reading single pixels is very ineffective.
To build sample code use command:

```
gcc –m32 –fpack-struct graph_io.c
```

You can see output produced by this program in the `output.txt` file (the last part presents the coordinates of patterns found by `FindPattern`). Additionally `result.bmp` contains all pattern occurrences inverted.
It would make sense to first experiment with memory access optimization in C and after checking that your approach works implement it in assembly language.