# Lab 3: MIPS assembler and parameters

## Objectives of this lab

In this lab we continue using the MIPS32 assembler. In particular, we will explore the utilization of parameters between functions and subroutines. Additionally, the students split the work into tasks describing this division in the Exercise 4.

## Description

In this lab you will implement the following set of functions:

1. Data print
2. String management
3. Functions to manage real and integer numbers
4. Functions to manage arrays

**NOTE** The functions MUST have the names indicated in the instructions. Pay special attention to uppercase and lowercase characters in the function names. In this assignment all the function names are low case.

## General considerations

### Passing parameters and return

Passing parameters to a subroutine will be done using the registers. The stack will only be use when needed. The parameters will be passed as follows:

- Non-floating point values will be passed using \$a0, \$a1, \$a2 and \$a3 registers. If more than four parameters are needed, then the stack will be used.
- Single precision floating point values will be passed using \$f12, \$f13, \$f14 and \$f15. If more than four parameters are needed, then the stack will be used.

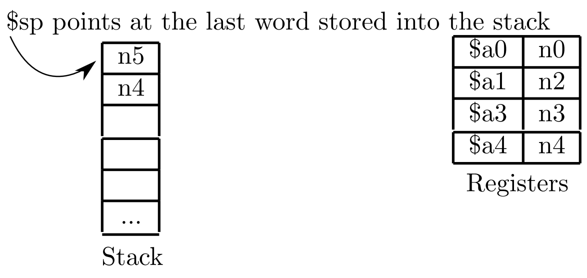
- Double precision floating point values will be passed using \$f12 and \$f14. If more than two parameters are needed, then the stack will be used.
- When returning non-floating point values registers \$v0 and \$v1 will be used. If more than two values are returned, then the stack will be used.
- When returning floating point values the register \$f0 will be used. If more than one value is returned, then the stack will be used.
- After finishing an invoked subroutine, the caller has to find the stack unmodified. This means that the caller has to use \$ra and pass the needed parameters to call the subroutine. When the subroutine finishes, this must restore the \$ra and the stack. E.g.:

```
        %%function A
        ...
        push parameters received by A and preserve registers
        push $ra
        push parameters (those not fitting into $a0, $a1,...)
        invoke function B
        pop parameters stored before calling B
        restore stack (recover $ra)
        return
        %%function B
        ...
        take parameters from the stack (when needed)
        push registers to preserve
        ...
        pop previously stored registers
        return
```

Example 1: for a function with the following signature in Java

```
int  function (int n0, int n1, int n2, int n3, int n4, int n5)
```

- Parameter n0 is passed in register \$a0.
- Parameter n1 is passed in register \$a1.
- Parameter n2 is passed in register \$a2.
- Parameter n3 is passed in register \$a3.
- Parameter n4 is pushed into the top of the stack.



- Parameter n5 is pushed below parameter n4 into the stack.
- The returned value is stored in $v0.

Graphically: The code inside the function must assumes that value n0 is stored in register $a0 and value n4 is the first position in the stack.

Example 2: for a function with the following signature in Java:

```
float  function (float n0, float n1, float n2, float n3, float n4, float n5)
```

- Parameter n0 is passed in register $f12.
- Parameter n1 is passed in register $f13.
- Parameter n2 is passed in register $f14.
- Parameter n3 is passed in register $f15.
- Parameter n4 is pushed into the top of the stack.
- Parameter n5 is pushed below parameter n4 into the stack.
- The returned value is stored in $f0.

## Passing arrays

Passing an array as a parameter independently of the array length employs two parameters:

- In the register $a0 is stored the array starting memory address
- In the registers $a1 is stored the array length

Example 3: for a function with the following signature in Java:

```
int function (float[] vector,  int n)
```

- Vector (starting memory address) is stored in $a0.
- Vector length n is stored in $a1.

- The result is returned in $v0.

Example 4: for a function with the following signature in Java

```
int function (char[] string)
```

- Vector (starting memory address) is stored in $a0.
- The result is returned in $v0.

When using strings of characters is not necessary to indicate the vector length. The string finishes with the end of string character (byte 0).

## Local variables

Local variables must keep their value during the function lifetime. Take into account the following recommendations:

- Local variables will always be assigned to registers $s, $t and $f for floating values.
- For variables not possible to be stored into registers, the stack will be used.

In MIPS we consider two kind of registers:

- Registers that preserves their value between calls to functions. These are the registers $s and $f20 and greater.
- Registers that do not guarantee their value between calls to functions. These are $t, $f4, $f6, $f8, $f10, $f16 and $f18.

## Exercise 1 (To be done during the lab session)

First, download from Aula Global the available lab material. Using the file **toLowerMod.s** code in assembler the function below. This program must work independently of the data declared in the .data section.

```
String string = "Test string for function 1";
char initial_char = 's';
char final_char = 'g';

public static void main(String[] args) {
        toLowerMod(string, initial_char, final_char);
        print_string(string);
}

void to_lower_mod(String string, char initial_char, char final_char){
        int position = 0;
        char character;

        while(position < string.length())
        {
                character = string.charAt(position);
                if(character < final_char && character > initial_char){
                        string[position] = '*';
                }
                position++;
        }
}

void print_string(String string){
        System.out.println(string);
}
```

**NOTE: during the lab session a modification in the previous exercise will be proposed. The attendance is MANDATORY.**

## Exercise 2

Implement the program **printf.s** that contain the following subroutines. In the provided lab material you can find the template to use.

```
String string_print = "This is the test string number %d. We can print integer
numbers such as %d, %d and %d. Additionally, we can print floats:
 %f, %f, %f, %f, %f and %f.";

int int1 = 1;
int int2 = 5;
int int3 = 9;
int int4 = 72;

float float1 = 5236.254f;
float float2 = 256.54f;
float float3 = -2568.2f;
float float4 = 25.92f;
float float5 = 6.4f;
float float6 = 2359.3358f;


public static void main(String[] args){
        printf(string_print, int1, int2, int3, int4,
        float1, float2, float3, float4, float5, float6);
}

void printf(String string_print, int int1, int int2, int int3, int int4,
float float1, float float2, float float3, float float4,
float float5, float float6)
{
        int counter = 0;
        int position = 0;
        char character;
        boolean percent = false;
        int auxInt = 0;
        float auxFloat = 0f;

        while(position < string_print.length())
        {
                character = string_print.charAt(position);
                if(!percent){
                        if(character == '%')
                                percent = true;
                        else
                                System.out.print(character);
                }else{
                        if(character == 'i'){
                                switch(counter){
                                        case 0:
                                                auxInt = int1;
                                                break;
                                        case 1:
                                                auxInt = int2;
                                                break;
                                        case 2:
                                                auxInt = int3;
                                                break;

                                                auxInt = int4;
                                                break;
                                }
                                print_int(auxInt);
                                counter++;
                                percent = false;
                        }else if(character == 'd'){
                                switch(counter){
                                case 4:
                                        auxFloat = float1;
                                        break;
                                case 5:
                                        auxFloat = float2;
                                        break;
                                case 6:
                                        auxFloat = float3;
                                        break;
                                case 7:
                                        auxFloat = float4;
                                        break;
                                case 8:
                                        auxFloat = float5;
                                        break;
                        case 9:
                                        auxFloat = float6;
                                        break;

                        }
                                print_float(auxFloat);
```

```
                                        counter++;
                                        percent = false;
                                }else{
                                        System.out.print('%');
                                        System.out.print(character);
                                        percent = false;
                                }
                        }
                        position++;
                }
}

void print_int(int integer)
{
        System.out.print(integer);
}

void print_float(float number)
{
        System.out.print(number);
}
```

The result for the previous code is:

```
This is the test string number 1. We can print integer numbers such as 5, 9
and 72. Additionally, we can print floats: 5236.254, 256.54, -2568.2, 25.92,
6.4 and 2359.3358.
```

The number of function parameters is constant. It will always receive four integers and six floats. The order to be used in the parameters load is the following:

- Parameter 1 (string) in a0
- Parameter 2 (first integer) in a1
- Parameter 3 (second integer) in a2
- Parameter 4 (third integer) in a3
- Parameter 5 (fourth integer) onto the stack
- Parameter 6 (first float) in f12
- Parameter 7 (second float) in f13
- Parameter 8 (third float) in f14
- Parameter 9 (fourth float) in f15

- Parameter 10 (fifth float) onto the stack
- Parameter 11 (sixth float) onto the stack

The code employed to test your solution will follow the mentioned order. The string and the value of the numbers can be modified, but not the number of parameters.

## Exercise 3

Implement the program **updateDiagonal.s** using the template contained in the provided lab materials. The final behavior of this program is to modify the diagonal of a matrix with the addition of the surrounding positions. Now we detailed the functions to be designed:

- `int get(int[][] array, int x, int y, int x_pos, int y_pos)`

  Given a position (x,y) of a matrix return the value stored in that position. Remember that matrices are stored as stored row by row. The Java code of this function is as follows:

```
int get(int[][] array, int x, int y, int x_pos, int y_pos){
        int number = 0;
        number = array[x_pos][y_pos];
        return number;
}
```

- `int neighbors(int[][] array, int x, int y, int x_pos, int y_pos)`

  This function returns the addition off all the neighbors for a given position.

```
int neighbors(int[][] array, int x, int y, int x_pos, int y_pos){
        int neighbors = 0;
        if(x_pos - 1 >= 0){
                neighbors += get(array, x, y, x_pos -1, y_pos);
        }
        if(y_pos - 1 >= 0){
                neighbors += get(array, x, y, x_pos, y_pos-1);
        }
        if(x_pos - 1 >= 0 && y_pos - 1 >= 0){
                neighbors += get(array, x, y, x_pos-1, y_pos-1);
        }
        if(x_pos + 1 < x){
                neighbors += get(array, x, y, x_pos+1, y_pos);
        }
        if(y_pos + 1 < y){
                neighbors += get(array, x, y, x_pos, y_pos+1);
        }
        if(x_pos + 1 < x && y_pos + 1 < y){
                neighbors += get(array, x, y, x_pos+1, y_pos+1);
        }
        if(x_pos - 1 >= 0 && y_pos + 1 < y){
                neighbors += get(array, x, y, x_pos-1, y_pos+1);
        }
        if(x_pos + 1 < x && y_pos - 1 >= 0){
                neighbors += get(array, x, y, x_pos+1, y_pos-1);
        }
        return neighbors;
}
```

- `void set(int[][] array, int number, int x, int y, int x_pos, int y_pos)`

  Given a position (x,y) of the matrix set the value of this position to the value stored in number.

  ```
  void set(int[][] array, int number, int x, int y, int x_pos, int y_pos){
          array[x_pos][y_pos] = number;
  }
  ```

- `void print(int[][] array, int x, int y)]`

  This function prints the matrix using the standard output.

  ```
  void print(int[][] array, int x, int y){
          int number = 0;
          for(int i = 0; i < x; i++){
                  for(int j = 0; j < y; j++){
                          number = get(array, x, y, i, j);
                          System.out.print(number);
                          System.out.print('\t');
                  }
                  System.out.print('\n');
          }
  }
  ```

- `void update_diagonal(int[][] array, int x, int y)`

  This function sets the value of the diagonal elements with the addition of their neighbors.

```
void update_diagonal(int[][] array, int x, int y){
        for(int i = 0; i < x && i < y; i++){
```

```
                int neighbors = neighbors(array, x, y, i, i);
                set(array, neighbors, x, y, i, i);
        }
}
```

The following example indicates how the previous code works.

```
int[] array = new int[]{1, 1, 1, 1,
                        1, 1, 1, 1,
                        1, 1, 1, 1,
                        1, 1, 1, 1};
int x = 4;
int y = 4;

public static void main(String[] args) {
        print(array, x, y);
        System.out.println();
        updateDiagonal(array, x, y);
        print(array, x, y);
}
```

And the output is:

```
 1  1  1  1
 1  1  1  1
 1  1  1  1
 1  1  1  1


3   1   1   1
1  10   1   1
1   1  17   1
1   1   1  19
```

**NOTE: the program must work independently of the data declared in the .data section.**

## Exercise 4

In this exercise the student will create a detailed workplan with the following points:

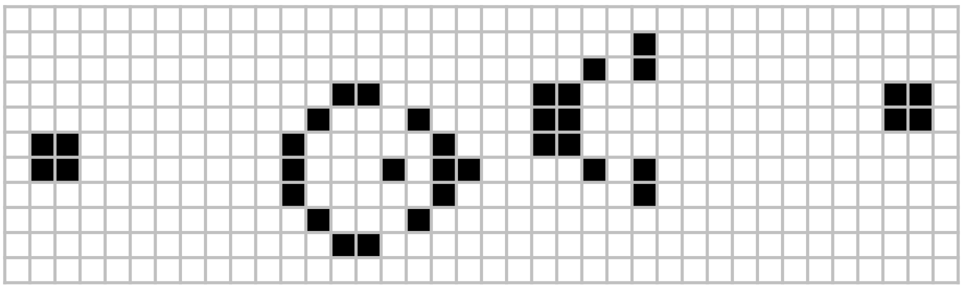
- Separation of tasks between the group members: by exercises, by functions, etc.
- Work time scheduling.
- Communication between the group members (meetings, documents, etc...).
- Tools employed for communication between the group members and work submission.
- Methods employed to merge the work done.
- Testing plan.
- How to evaluate the work plan.

**NOTE: it is mandatory to submit this exercise to pass the lab. This exercise will be evaluated as passed or non-passed.**

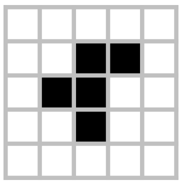
## Exercise 5

Based on the routines developed in the exercise 3 the student will develop a game of life using the template **conway.s** included in the lab material. The following URL describes the rules of the game:

`http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life`



The figure above is an example of game of life called Gosper glider gun. This input matrix generates infinite planners as the one showed below.



In this case each cell is one position in the array. A cell is alive if the matrix position is 1. The cell is dead if the value is 0. A dead cell will come to life if three cells around are also alive. One cell will be alive if and only if 2 or 3 cells around are also alive. Otherwise it will die. In order to code this program the following functions has to be done:

- malloc(int x, int y) This function allocates memory dynamically. Receives the dimension of the matrix to create. The function returns the address of the matrix.

```
int[] malloc(int x, int y){
        int[] array = new int[x*y];
        return array;
}
```

- copy_array (int[] src_array, int[] dest_array, int x, int y)

  This function copies one array into another.

```
void copy_array(int[] src_array,int [] dest_array, int x, int y){
        int number = 0;
        for(int i = 0; i < x; i++){
                for (int j = 0; j < y; j++){
                        number = get(src_array, x, y, i, j);
                        set(dest_array, number, x, y, i, j);
                }
        }
}
```

- clean_array(int[] array, int x, int y)

  This function cleans the memory area where the matrix was allocated. Each previous array position is set to 0.

```
void clean_array(int[] array, int x, int y)        {
        for(int i = 0; i < x; i++){
                for (int j = 0; j < y; j++){
                        set(array, 0, x, y, i, j);
                }
        }
}
```

- game_of_life (int[] array, int x, int y, int iterations)

  This function iterates the game of life for a given initial matrix, its dimensions, and the number of iterations. In each iteration the new state will be stored in an auxiliary variable.

```
void game_of_life(int[] array, int x, int y, int iterations){
        int[] array_aux = malloc(x, y);
        int neighbors = 0;
        int cell = 0;

        for(int it = 0; it < iterations; it++){
                clean_array(array_aux, x, y);
                for(int i = 0; i < x; i++){
                        for(int j = 0; j < y; j++){
                                cell = get(array, x, y, i, j);
                                neighbors = neighbors(array, x, y ,i, j);
                                if(cell == 0){
                                        if(neighbors == 3){
                                                set(array_aux, 1, x, y,i, j);
                                        }else{
                                                set(array_aux, 0, x, y,i, j);
                                        }
                                }else{
                                        if(neighbors == 2||neighbors == 3){
                                                set(array_aux, 1, x, y,i, j);
                                        }else{
                                                set(array_aux, 0, x, y,i, j);
                                        }
                                }
                        }
                }
                copy_array(array_aux, array, x, y);
        }
}
```

**NOTE: the program must work independently of the declared .data section.**

## Evaluation

This lab will be evaluated in three parts :

1. **Presential part (2 points)**. This submission will be done during the lab session. Only exercise 1 must be submitted in this part.
2. **Mandatory part (5 points)**. This part includes exercises 2, 3 and 4 with the final document. The exercises are graded as follows:
   (a) Exercise 2 (2 points)
   (b) Exercise 3 (2 points)
   (c) Exercise 4 (Passed or not passed)
   (d) Final document (1 point)
3. **Optional part (3 points)**. Exercise 5 is optional and will be grades as follows:
   (a) Exercise 5 (2.5 points)
   (b) Explanations in the final document (0.5 points)

## Submission

The submission will be as follows:

- **toLowerMod.s** will be submitted through aula global during the lab session. JUST ONE ZIPPED file with name **ec_p3_e1_A...A_B...B.pdf** with A...A and B...B the student identifiers of the group members.
- The remainder exercises and questions will be submitted a zip file with name **ec_p3_e2_A...A_B...B.zip** before **December 14th** at 23:55 through Aula Global. This zipped file must contain the assembly source code (*printf.s*, *updateDiagonals.s*, *conway.s*) and the Document.pdf file with the answers for the formulated questions, the explanation of your code with the explanations you may consider useful (flow diagrams, algorithms, etc) and the taken design decisions. NO DOCUMENT WILL BE REVIEWED if it does not include at least:
    1. Front page indicating authors and their student numbers.
    2. Index of contents.
    3. Description of the implemented methods.
    4. Done tests and their results.
    5. Conclusions.
    6. Justified margins and numerated pages.

The final document is fundamental to get a good mark in your lab work. It is also necessary to submit the correct code, with the requested method names. The length of the final document MUST NOT exceed 10 pages (index and front page included). If the final document does not fulfill the mentioned requirements, the lab will be failed.

- The submission will ONLY be done through AULA GLOBAL.
- The version to be reviewed is the last submitted one.

## Rules

1. The code MUST compile and MUST work as expected.
2. Non-commented code is considered void.
3. The submission MUST be done through AULA GLOBAL. No alternative methods can be used.
4. The submitted code and final document MUST be original work. If cheating is detected, all the students involved in the copy will fail the lab.