# Week 07 Laboratory Sample Solutions

## Objectives

- learning how function calls are implemented
- practicing MIPS memory access
- practicing calculating array indices
- practicing using MIPS control instructions (branch)
- practicing running MIPS programs with spim, xspim or qtspim

---

## Preparation

Before the lab you should re-read the relevant lecture slides and their accompanying examples.

---

## Getting Started

Create a new directory for this lab called `lab07`, change to this directory, and fetch the provided code for this week by running these commands:

```
$ mkdir lab07
$ cd lab07
$ 1521 fetch lab07
```

Or, if you're not working on CSE, you can download the provided code as a zip file or a tar file.

---

EXERCISE — INDIVIDUAL:

## Create An addi Instruction

Your task is to add code to this function in **addi.c**:

```
// return the MIPS opcode for addi $t,$s, i
uint32_t addi(int t, int s, int i) {

    return 42; // REPLACE WITH YOUR CODE

}
```

The function `addi` is given the operands for a MIPS **addi** instruction . Add code so that it returns the opcode for that instruction.

The assignment 1 specifcation provides the general bit pattern for an **addi** instruction.

`1521 spim2hex` can be used to print the opcode for a particular **addi** instruction, for example:

```
$ echo 'addi $17 $19 -3'|1521 spim2hex
2271fffd
$ ./addi 17 19 -3
addi(17, 19, -3) returned 0x2271fffd
$ echo 'addi $9 $27 42'|1521 spim2hex
2369002a
$ ./addi 9 27 42
2369002a
```

Use *make* to build your code:

```
$ make    # or 'make addi'
```

**Assumptions/Limitations/Clarifications**

You may define and call your own functions if you wish.

You are not permitted to change the `main` function you have been given, or to change `addi`' prototype (its return type and argument types).

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 1521 autotest addi
```

When you are finished working on this exercise, you must submit your work by running `give`:

```
$ give cs1521 lab07_addi addi.c
```

You must run `give` before **Wednesday 04 November 21:00** to obtain the marks for this lab exercise. Note that this is an individual exercise, the work you submit with `give` must be entirely your own.

Sample solution for `addi.c`

```c
// Sample solution for COMP1521 lab exercises
//
// generate the opcode for an addi instruction

#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <assert.h>

#include "addi.h"

// return the MIPS opcode for addi $t,$s, i
uint32_t addi(int t, int s, int i) {
    return          0x20000000 |
           ((uint32_t)s) << 21 |
           ((uint32_t)t) << 16 |
           (i & 0xFFFF);
}
```

---

EXERCISE — INDIVIDUAL:

## MIPS 2d array

In the files for this lab, you have been given `lookup.s`, a MIPS assembler program that reads 2 numbers and then prints 42.

Add code to `lookup.s` to make it equivalent to this C program:

```c
// Read 2 numbers and use them as indices into a 2d-array

#include <stdio.h>

int array[42][24] = {
    { 9, 4, 3, 2, 5, 1, 1, 4, 3, 1, 2, 6, 7, 5, 6, 2, 8, 1, 8, 3, 4, 1, 1, 1 },
    { 7, 3, 9, 6, 6, 2, 4, 8, 6, 8, 1, 9, 8, 2, 9, 5, 9, 8, 9, 9, 2, 3, 1, 1 },
    { 1, 4, 6, 5, 4, 2, 9, 5, 7, 9, 5, 6, 4, 1, 6, 9, 6, 1, 9, 3, 8, 3, 1, 2 },
    { 3, 3, 3, 9, 7, 3, 7, 1, 3, 2, 3, 7, 7, 3, 2, 5, 8, 1, 4, 2, 4, 5, 9, 4 },
    { 5, 7, 6, 9, 4, 2, 4, 9, 4, 7, 9, 2, 8, 1, 6, 2, 7, 4, 9, 7, 1, 9, 3, 5 },
    { 8, 3, 8, 9, 3, 4, 6, 2, 4, 1, 3, 3, 2, 1, 2, 4, 2, 7, 8, 6, 8, 6, 9, 9 },
    { 9, 5, 8, 9, 7, 5, 6, 6, 2, 9, 5, 1, 1, 8, 6, 8, 3, 4, 1, 1, 5, 9, 5, 2 },
    { 3, 2, 4, 1, 4, 8, 2, 8, 7, 6, 7, 8, 8, 3, 8, 2, 6, 5, 5, 5, 5, 9, 5, 3 },
    { 7, 1, 3, 9, 8, 8, 6, 3, 1, 7, 6, 5, 6, 9, 3, 8, 1, 5, 7, 6, 7, 7, 5, 6 },
    { 4, 6, 5, 7, 4, 1, 4, 7, 3, 5, 5, 7, 9, 6, 8, 4, 3, 1, 9, 9, 2, 6, 8, 9 },
    { 2, 3, 8, 5, 8, 8, 7, 1, 8, 1, 1, 8, 2, 2, 3, 9, 7, 6, 7, 9, 3, 2, 6, 5 },
    { 1, 4, 7, 4, 7, 7, 7, 7, 9, 9, 8, 9, 5, 5, 3, 3, 9, 5, 8, 7, 7, 6, 1, 7 },
    { 5, 3, 8, 7, 5, 6, 1, 9, 5, 6, 3, 3, 5, 9, 9, 5, 4, 1, 3, 8, 1, 1, 1, 4 },
    { 9, 8, 1, 7, 5, 1, 7, 4, 9, 7, 4, 8, 2, 5, 9, 3, 6, 3, 6, 3, 2, 7, 3, 2 },
    { 1, 6, 1, 4, 2, 9, 6, 1, 3, 2, 5, 7, 3, 9, 4, 4, 6, 5, 9, 8, 4, 5, 1, 4 },
    { 7, 7, 7, 2, 1, 6, 1, 3, 9, 4, 4, 6, 6, 6, 3, 9, 3, 8, 2, 8, 8, 4, 8, 7 },
    { 7, 8, 7, 9, 3, 5, 7, 1, 1, 4, 1, 4, 9, 6, 7, 3, 8, 5, 1, 7, 9, 2, 2, 2 },
    { 2, 4, 6, 5, 7, 3, 4, 6, 1, 7, 2, 5, 1, 7, 1, 2, 9, 6, 7, 8, 5, 4, 5, 7 },
    { 2, 4, 4, 9, 2, 8, 1, 9, 5, 9, 5, 9, 8, 3, 4, 7, 6, 7, 5, 2, 9, 9, 5, 5 },
    { 8, 4, 2, 6, 3, 8, 8, 3, 6, 3, 2, 4, 5, 1, 8, 6, 6, 4, 5, 8, 4, 6, 8, 5 },
    { 7, 7, 9, 8, 4, 1, 1, 3, 8, 8, 7, 6, 3, 8, 1, 2, 2, 4, 4, 5, 3, 5, 9, 9 },
    { 5, 7, 1, 7, 5, 5, 8, 1, 4, 6, 5, 7, 5, 9, 3, 7, 4, 8, 6, 4, 1, 6, 7, 1 },
    { 4, 5, 3, 3, 1, 2, 5, 3, 1, 5, 7, 6, 6, 2, 8, 8, 8, 3, 6, 3, 1, 2, 6, 3 },
    { 9, 5, 3, 4, 7, 2, 9, 9, 8, 6, 2, 5, 9, 3, 1, 8, 6, 9, 6, 3, 3, 2, 3, 3 },
    { 8, 6, 5, 3, 3, 7, 6, 3, 3, 9, 1, 4, 7, 5, 1, 6, 5, 1, 6, 8, 8, 1, 9, 7 },
    { 4, 7, 5, 9, 1, 7, 6, 9, 5, 2, 3, 7, 3, 8, 8, 3, 9, 8, 5, 6, 1, 6, 6, 9 },
    { 2, 8, 6, 9, 3, 3, 6, 9, 4, 5, 2, 6, 3, 8, 3, 9, 6, 7, 6, 5, 6, 8, 2, 6 },
    { 4, 8, 6, 4, 5, 3, 9, 4, 3, 4, 7, 9, 9, 4, 5, 8, 6, 6, 3, 4, 7, 1, 3, 4 },
    { 7, 4, 6, 7, 1, 9, 6, 2, 8, 4, 5, 6, 7, 6, 4, 1, 6, 3, 1, 2, 5, 9, 2, 1 },
    { 2, 8, 9, 1, 6, 5, 1, 7, 2, 3, 3, 5, 4, 8, 6, 1, 9, 8, 5, 8, 1, 4, 4, 7 },
    { 8, 8, 2, 9, 9, 4, 8, 8, 9, 2, 6, 4, 2, 8, 1, 2, 3, 3, 9, 5, 3, 1, 1, 1 },
    { 3, 9, 5, 7, 7, 9, 7, 3, 4, 2, 1, 8, 6, 3, 6, 9, 3, 3, 4, 2, 5, 1, 2, 3 },
    { 4, 4, 6, 4, 5, 8, 1, 7, 4, 4, 6, 6, 9, 7, 9, 4, 3, 6, 6, 4, 9, 8, 2, 6 },
    { 3, 8, 2, 2, 7, 4, 3, 8, 7, 4, 1, 6, 6, 2, 3, 5, 2, 1, 8, 4, 6, 4, 8, 6 },
    { 5, 2, 5, 6, 5, 9, 3, 3, 8, 1, 3, 8, 2, 9, 2, 8, 9, 7, 2, 7, 5, 5, 7, 7 },
    { 2, 7, 6, 4, 3, 2, 1, 4, 6, 3, 7, 5, 7, 7, 5, 6, 4, 6, 8, 2, 9, 3, 6, 1 },
    { 6, 4, 4, 6, 1, 4, 2, 6, 3, 7, 9, 9, 4, 4, 2, 1, 8, 1, 4, 4, 2, 7, 4, 9 },
    { 3, 8, 5, 2, 3, 9, 2, 4, 8, 9, 3, 3, 6, 2, 3, 3, 1, 8, 5, 8, 8, 5, 1, 9 },
    { 1, 5, 8, 1, 4, 9, 2, 4, 9, 5, 7, 6, 7, 4, 8, 9, 1, 3, 8, 6, 4, 4, 9, 9 },
    { 5, 6, 7, 8, 3, 2, 9, 1, 1, 7, 7, 6, 9, 7, 7, 7, 8, 8, 3, 3, 8, 9, 9, 1 },
    { 8, 2, 5, 9, 1, 1, 7, 6, 3, 6, 7, 7, 7, 2, 4, 5, 5, 2, 1, 1, 1, 7, 4, 3 },
    { 8, 9, 4, 5, 4, 6, 2, 5, 3, 7, 5, 1, 6, 7, 2, 8, 5, 6, 2, 2, 1, 7, 6, 2 },
};

int main(void) {
    int x, y;
    printf("Enter x: ");
    scanf("%d", &x);
    printf("Enter y: ");
    scanf("%d", &y);

    printf("%d\n", array[x][y]);

    return 0;
}
```

In other words, it should read 2 numbers and use them as array indices to print a value from a 2 dimensional array. For example:

```
$ 1521 spim -f lookup.s
Enter x: 5
Enter y: 8
4
$ 1521 spim -f lookup.s
Enter x: 41
Enter y: 23
2
$ 1521 spim -f lookup.s
Enter x: 0
Enter y: 0
9
```

**HINT:**

The MIPS you given already has suitable directives to create an array equivalent to the one in the C program.

### Assumptions/Limitations/Clarifications

You can assume both numbers read are valid array indices.

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 1521 autotest lookup
```

When you are finished working on this exercise, you must submit your work by running `give`:

```
$ give cs1521 lab07_lookup lookup.s
```

You must run `give` before **Wednesday 04 November 21:00** to obtain the marks for this lab exercise. Note that this is an individual exercise, the work you submit with `give` must be entirely your own.

Sample solution for `lookup.s`

```
# Read 2 numbers and use them as indices into a 2d-array
# x in $t0, y in $t1
# $t2..$t7 used for temporary values
main:
    la $a0, prompt_x    # printf("Enter x: ");
    li $v0, 4
    syscall

    li $v0, 5           # scanf("%d", &x);
    syscall             #
    move $t0, $v0

    la $a0, prompt_y    # printf("Enter y: ");
    li $v0, 4
    syscall

    li $v0, 5           # scanf("%d", &y);
    syscall             #
    move $t1, $v0

    mul $t3, $t0, 24    # calculate &array[x][y]
    add $t4, $t3, $t1
    mul $t5, $t4, 4
    la $t6, array
    add $t7, $t6, $t5

    lw $a0, 0($t7)      # printf("%d", array[x][y])
    li $v0, 1
    syscall

    li   $a0, '\n'      # printf("%c", '\n');
    li   $v0, 11
    syscall

    li $v0, 0           # return 0
    jr $31

.data
prompt_x:
    .asciiz "Enter x: "
prompt_y:
    .asciiz "Enter y: "
array:
    .word 9 4 3 2 5 1 1 4 3 1 2 6 7 5 6 2 8 1 8 3 4 1 1 1
    .word 7 3 9 6 6 2 4 8 6 8 1 9 8 2 9 5 9 8 9 9 2 3 1 1
    .word 1 4 6 5 4 2 9 5 7 9 5 6 4 1 6 9 6 1 9 3 8 3 1 2
    .word 3 3 3 9 7 3 7 1 3 2 3 7 7 3 2 5 8 1 4 2 4 5 9 4
    .word 5 7 6 9 4 2 4 9 4 7 9 2 8 1 6 2 7 4 9 7 1 9 3 5
    .word 8 3 8 9 3 4 6 2 4 1 3 3 2 1 2 4 2 7 8 6 8 6 9 9
    .word 9 5 8 9 7 5 6 6 2 9 5 1 1 8 6 8 3 4 1 1 5 9 5 2
    .word 3 2 4 1 4 8 2 8 7 6 7 8 8 3 8 2 6 5 5 5 5 9 5 3
    .word 7 1 3 9 8 8 6 3 1 7 6 5 6 9 3 8 1 5 7 6 7 7 5 6
    .word 4 6 5 7 4 1 4 7 3 5 5 7 9 6 8 4 3 1 9 9 2 6 8 9
    .word 2 3 8 5 8 8 7 1 8 1 1 8 2 2 3 9 7 6 7 9 3 2 6 5
    .word 1 4 7 4 7 7 7 7 9 9 8 9 5 5 3 3 9 5 8 7 7 6 1 7
    .word 5 3 8 7 5 6 1 9 5 6 3 3 5 9 9 5 4 1 3 8 1 1 1 4
    .word 9 8 1 7 5 1 7 4 9 7 4 8 2 5 9 3 6 3 6 3 2 7 3 2
    .word 1 6 1 4 2 9 6 1 3 2 5 7 3 9 4 4 6 5 9 8 4 5 1 4
    .word 7 7 7 2 1 6 1 3 9 4 4 6 6 6 3 9 3 8 2 8 8 4 8 7
    .word 7 8 7 9 3 5 7 1 1 4 1 4 9 6 7 3 8 5 1 7 9 2 2 2
    .word 2 4 6 5 7 3 4 6 1 7 2 5 1 7 1 2 9 6 7 8 5 4 5 7
    .word 2 4 4 9 2 8 1 9 5 9 5 9 8 3 4 7 6 7 5 2 9 9 5 5
    .word 8 4 2 6 3 8 8 3 6 3 2 4 5 1 8 6 6 4 5 8 4 6 8 5
    .word 7 7 9 8 4 1 1 3 8 8 7 6 3 8 1 2 2 4 4 5 3 5 9 9
    .word 5 7 1 7 5 5 8 1 4 6 5 7 5 9 3 7 4 8 6 4 1 6 7 1
    .word 4 5 3 3 1 2 5 3 1 5 7 6 6 2 8 8 8 3 6 3 1 2 6 3
    .word 9 5 3 4 7 2 9 9 8 6 2 5 9 3 1 8 6 9 6 3 3 2 3 3
    .word 8 6 5 3 3 7 6 3 3 9 1 4 7 5 1 6 5 1 6 8 8 1 9 7
    .word 4 7 5 9 1 7 6 9 5 2 3 7 3 8 8 3 9 8 5 6 1 6 6 9
    .word 2 8 6 9 3 3 6 9 4 5 2 6 3 8 3 9 6 7 6 5 6 8 2 6
    .word 4 8 6 4 5 3 9 4 3 4 7 9 9 4 5 8 6 6 3 4 7 1 3 4
    .word 7 4 6 7 1 9 6 2 8 4 5 6 7 6 4 1 6 3 1 2 5 9 2 1
    .word 2 8 9 1 6 5 1 7 2 3 3 5 4 8 6 1 9 8 5 8 1 4 4 7
    .word 8 8 2 9 9 4 8 8 9 2 6 4 2 8 1 2 3 3 9 5 3 1 1 1
    .word 3 9 5 7 7 9 7 3 4 2 1 8 6 3 6 9 3 3 4 2 5 1 2 3
```

```
.word 4 4 6 4 5 8 1 7 4 4 6 6 9 7 9 4 3 6 6 4 9 8 2 6
.word 3 8 2 2 7 4 3 8 7 4 1 6 6 2 3 5 2 1 8 4 6 4 8 6
.word 5 2 5 6 5 9 3 3 8 1 3 8 2 9 2 8 9 7 2 7 5 5 7 7
.word 2 7 6 4 3 2 1 4 6 3 7 5 7 7 5 6 4 6 8 2 9 3 6 1
.word 6 4 4 6 1 4 2 6 3 7 9 9 4 4 2 1 8 1 4 4 2 7 4 9
.word 3 8 5 2 3 9 2 4 8 9 3 3 6 2 3 3 1 8 5 8 8 5 1 9
.word 1 5 8 1 4 9 2 4 9 5 7 6 7 4 8 9 1 3 8 6 4 4 9 9
.word 5 6 7 8 3 2 9 1 1 7 7 6 9 7 7 7 8 8 3 3 8 9 9 1
.word 8 2 5 9 1 1 7 6 3 6 7 7 7 2 4 5 5 2 1 1 1 7 4 3
.word 8 9 4 5 4 6 2 5 3 7 5 1 6 7 2 8 5 6 2 2 1 7 6 2
```

---

EXERCISE — INDIVIDUAL:

## MIPS Sieve

In the files for this lab, you have been given `sieve.s`.

Add code to `sieve.s` to make it equivalent to this C program:

```c
// Sieve of Eratosthenes
// https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes
#include <stdio.h>
#include <stdint.h>

uint8_t prime[1000];

int main(void) {
    int i = 0;
    while (i < 1000) {
        prime[i] = 1;
        i++;
    }

    i = 2;
    while (i < 1000) {
        if (prime[i]) {
            printf("%d\n", i);
            int j = 2 * i;
            while (j < 1000) {
                prime[j] = 0;
                j = j + i;
            }
        }
        i++;
    }
    return 0;
}
```

Use the space in the data area to store the array `prime`. For example:

```
$ 1521 spim -f sieve.s
2
3
5
7
11
13
17
19
23
...
971
977
983
991
997
```

**HINT:**

Use `lb` and `sb` instruction to access array `prime`.

**NOTE:**

You must implement the algorithm in `sieve.c`. You are not permitted to use another algorithm to print prime numbers.

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 1521 autotest sieve
```

When you are finished working on this exercise, you must submit your work by running `give`:

```
$ give cs1521 lab07_sieve sieve.s
```

You must run `give` before **Wednesday 04 November 21:00** to obtain the marks for this lab exercise. Note that this is an individual exercise, the work you submit with `give` must be entirely your own.

Sample solution for `sieve.s`

```
# i in register $t0
# j in register $t1
# registers $t2, $t3, $t4 used to hold temporary results

main:
    li $t0, 0           # i = 0;
loop0:
    bge  $t0, 1000, end0 # while (i < 1000) {
    la   $t2, prime      #   calculate &prime[i]
    add  $t3, $t2, $t0   #
    li   $t4, 1
    sb   $t4, ($t3)      #   prime[i] = 1

    add $t0, $t0, 1      #   i++;
    b loop0              # }
end0:

    li $t0, 2           # i = 2;
loop1:
    bge $t0, 1000, done  # while (i < 1000) {

    la $t2, prime        #   calculate &prime[i]
    add $t3, $t2, $t0    #
    lb $t4, ($t3)        #   load prime[i] into $t3
    bne $t4, 1, end2     #   if (prime[i]) {

    move $a0, $t0        #     printf("%d", i);
    li $v0, 1
    syscall

    li   $a0, '\n'       #     printf("%c", '\n');
    li   $v0, 11
    syscall

    mul $t1, $t0, 2      #     j = 2 * i;
loop2:
    bge $t1, 1000, end2  #     while (j < 1000) {
    la $t2, prime        #       calculate &prime[j]
    add $t3, $t2, $t1    #
    sb $0, ($t3)         #        prime[j] = 0

    add $t1, $t1, $t0    #       j = j + i;
    b loop2              #     }
end2:                    #   }
    add $t0, $t0, 1      #   i++;
    b loop1              # }

done:
    li $v0, 0           # return 0
    jr $31

    .data
prime:
    .space 1000
```

EXERCISE — INDIVIDUAL:

# MIPS factorial

In the files for this lab, you have been given `factorial.s`. Add code to make it equivalent to this C program:

```c
// Recursive factorial function
// n < 1 yields n! = 1

#include <stdio.h>

int factorial(int);

int main(void) {
    int n = 0;
    printf("Enter n: ");
    scanf("%d", &n);
    int f = factorial(n);
    printf("%d! = %d\n", n, f);
    return 0;
}

int factorial(int n) {
    int result;
    if (n > 1) {
        result = n * factorial(n - 1);
    } else {
        result = 1;
    }
    return result;
}
```

For example:

```
$ 1521 spim -f factorial.s
Enter n: 5
5! = 120
$ 1521 spim -f factorial.s
Enter n: 7
7! = 5040
$ 1521 spim -f factorial.s
Enter n: 1
1! = 1
```

**NOTE:**

You must implement the code in `factorial.c` as a recursive function.

You may not use another algorithm to calculate or print factorials.

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 1521 autotest factorial
```

When you are finished working on this exercise, you must submit your work by running `give`:

```
$ give cs1521 lab07_factorial factorial.s
```

You must run `give` before **Wednesday 04 November 21:00** to obtain the marks for this lab exercise. Note that this is an individual exercise, the work you submit with `give` must be entirely your own.

Sample solution for `factorial.s`

```
# Recursive factorial function
# n < 1 yields n! = 1
# $s0 is used for n
# we use an s register because the convention is their value
# is preserved across function calls
# f is in $t0

main:
    addi $sp, $sp, -8  # create stack frame
    sw   $ra, 4($sp)   # save return address
    sw   $s0, 0($sp)   # save $s0

    li   $s0, 0
    la   $a0, msg1
    li   $v0, 4
    syscall            # printf(Enter n: ")

    li    $v0, 5
    syscall            # scanf("%d", &n)
    move  $s0, $v0

    move  $a0, $s0     # factorial(n)
    jal   factorial    #
    move  $t0, $v0     #

    move  $a0, $s0
    li    $v0, 1
    syscall            # printf ("%d", n)

    la    $a0, msg2
    li    $v0, 4
    syscall            # printf("! = ")

    move  $a0, $t0
    li    $v0, 1
    syscall            # printf ("%d", f)

    li   $a0, '\n'     # printf("%c", '\n');
    li   $v0, 11
    syscall

                       # clean up stack frame
    lw   $s0, 0($sp)   # restore $s0
    lw   $ra, 4($sp)   # restore $ra
    addi $sp, $sp, 8   # restore sp

    li  $v0, 0         # return 0
    jr  $ra

    .data
msg1:   .asciiz "Enter n: "
msg2:   .asciiz "! = "


# $s0 is used for n
# we use an s register because the convention is their value
# is preserved across function calls
# result is in $t0

    .text
factorial:
    addi $sp, $sp, -8  # create stack frame
    sw   $ra, 4($sp)   # save return address
    sw   $s0, 0($sp)   # save $s0

    move $s0, $a0

    ble  $s0, 1, else  # if (n > 1) {
    addi $a0, $s0, -1  #   result = n * fac(n - 1);
    jal  factorial     #
    mul  $t0, $v0, $s0 #
    j    end
else:                  # } else {
    li   $t0, 1        #   result = 1
end:                   # }
```

```
    move $v0, $t0

    lw   $s0, 0($sp)   # restore $s0
    lw   $ra, 4($sp)   # restore $ra
    addi $sp, $sp, 8   # restore sp

    jr  $ra
```

---

CHALLENGE EXERCISE — INDIVIDUAL:

## MIPS Big Factorials

Create a MIPS program `big_factorial.s` which will calculate arbitrarily large factorials. For example:

```
$ 1521 spim -f big_factorial.s
Enter n: 42
42! = 1405006117752879898543142606244511569936384000000000
$ 1521 spim -f big_factorial.s
Enter n: 112
112! =
19745068572210740235368203727599248834127786803497533779665629509490285896977181144089422435502777936659795733823785363827230
```

**NOTE:**

You are not permitted to use floating point arithmetic (`float` or `double`).

Your MIPS code must calculate the factorial. You cannot look up precalculated values.

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 1521 autotest big_factorial
```

When you are finished working on this exercise, you must submit your work by running `give`:

```
$ give cs1521 lab07_big_factorial big_factorial.s
```

You must run `give` before **Wednesday 04 November 21:00** to obtain the marks for this lab exercise. Note that this is an individual exercise, the work you submit with `give` must be entirely your own.

---

## Submission

When you are finished each exercises make sure you submit your work by running `give`.

You can run `give` multiple times. Only your last submission will be marked.

Don't submit any exercises you haven't attempted.

If you are working at home, you may find it more convenient to upload your work via give's web interface.

Remember you have until **Wed Nov 4 21:00:00 2020** to submit your work.

You cannot obtain marks by e-mailing your code to tutors or lecturers.

You check the files you have submitted here.

Automarking will be run by the lecturer several days after the submission deadline, using test cases different to those `autotest` runs for you. (Hint: do your own testing as well as running `autotest`.)

After automarking is run by the lecturer you can view your results here. The resulting mark will also be available via give's web interface.

## Lab Marks

When all components of a lab are automarked you should be able to view the the marks via give's web interface or by running this command on a CSE machine:

```
$ 1521 classrun -sturec
```

---