# Week 04 Laboratory Exercises

## Objectives

- learning to run MIPS programs with spim, xspim or qtspim
- understanding MIPS I/O (syscalls)
- understanding MIPS control instructions (branch)

## 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 `lab04`, change to this directory, and fetch the provided code for this week by running these commands:

```
$ mkdir lab04
$ cd lab04
$ 1521 fetch lab04
```

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

---

EXERCISE — INDIVIDUAL:
## Do You MIPS me?

Write a MIPS assembler program `bad_pun.s`, which is equivalent to this C program:

```c
// A simple C program that attempts to be punny

#include <stdio.h>

int main(void) {
    printf("I MIPS you!\n");

    return 0;
}
```

For example:

```
$ 1521 spim -f bad_pun.s
I MIPS you!
```

> **HINT:**
>
> The [i_love_mips.s](#) lecture example would make a good starting point.

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

```
$ 1521 autotest bad_pun
```

### Autotest Results

100% of 102 students who have autotested bad_pun.s so far, passed the autotest test.

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

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

```
$ give cs1521 lab04_bad_pun bad_pun.s
```

You must run `give` before **Friday 03 July 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.

---

EXERCISE — INDIVIDUAL:
# MIPS Grading

In the files for this lab, you have been given `grade.s`, a MIPS assembler program which reads a number and always prints **FL**:

```
$ 1521 spim -f grade.s
Loaded: /home/cs1521/share/spim/exceptions.s
Enter a mark: 100
FL
```

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

```c
// read a mark and print the corresponding UNSW grade

#include <stdio.h>

int main(void) {
    int mark;

    printf("Enter a mark: ");
    scanf("%d", &mark);

    if (mark < 50) {
        printf("FL\n");
    } else if (mark < 65) {
        printf("PS\n");
    } else if (mark < 75) {
        printf("CR\n");
    } else if (mark < 85) {
        printf("DN\n");
    } else {
        printf("HD\n");
    }

    return 0;
}
```

For example:

```
$ 1521 spim -f grade.s
Loaded: /home/cs1521/share/spim/exceptions.s
Enter a mark: 42
FL
$ 1521 spim -f grade.s
Loaded: /home/cs1521/share/spim/exceptions.s
Enter a mark: 72
CR
$ 1521 spim -f grade.s
Loaded: /home/cs1521/share/spim/exceptions.s
Enter a mark: 89
HD
```

> **HINT:**
>
> Make sure you understand the [odd_even.s](#) lecture example.

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

```
$ 1521 autotest grade
```

## Autotest Results

100% of 84 students who have autotested grade.s so far, passed all autotest tests.

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

```
$ give cs1521 lab04_grade grade.s
```

You must run `give` before **Friday 03 July 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.

---

> EXERCISE — INDIVIDUAL:
> # MIPS Counting

In the files for this lab, you have been given a MIPS assembler program `count.s`, which reads a number and prints 42:

```
$ 1521 spim -f count.s
Loaded: /home/cs1521/share/spim/exceptions.s
Enter a number: 13
42
```

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

```c
// read a number n and print the integers 1..n one per line

#include <stdio.h>

int main(void) {
    int number, i;

    printf("Enter number: ");
    scanf("%d", &number);

    i = 1;
    while (i <= number) {
        printf("%d\n", i);
        i = i + 1;
    }

    return 0;
}
```

For example:

```
$ 1521 spim -f count.s
Loaded: /home/cs1521/share/spim/exceptions.s
Enter number: 4
1
2
3
4
$ 1521 spim -f count.s
Loaded: /home/cs1521/share/spim/exceptions.s
Enter number: 10
1
2
3
4
5
6
7
8
9
10
```

> HINT:
>
> Make sure you understand the [print10.s](print10.s) lecture example.
>
> Started by choosing which register you will use for each variable.

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

```
$ 1521 autotest count
```

## Autotest Results

95% of 77 students who have autotested count.s so far, passed all autotest tests.

- 95% passed test *0 1 2 3 4 5*

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

```
$ give cs1521 lab04_count count.s
```

You must run `give` before **Friday 03 July 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.

---

EXERCISE — INDIVIDUAL:
# MIPS 7-Eleven

In the files for this lab, you have been given `seven_eleven.s`, a MIPS assembler program which reads a number and prints 42:

```
$ 1521 spim -f seven_eleven.s
Loaded: /home/cs1521/share/spim/exceptions.s
Enter a number: 13
42
```

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

```c
// Read a number and print positive multiples of 7 or 11 < n

#include <stdio.h>

int main(void) {
    int number, i;

    printf("Enter number: ");
    scanf("%d", &number);

    i = 1;
    while (i < number) {
        if (i % 7 == 0 || i % 11 == 0) {
            printf("%d\n", i);
        }
        i = i + 1;
    }

    return 0;
}
```

For example:

```
$ 1521 spim -f seven_eleven.s
Loaded: /home/cs1521/share/spim/exceptions.s
Enter number: 15
7
11
14
$ 1521 spim -f seven_eleven.s
Loaded: /home/cs1521/share/spim/exceptions.s
Enter number: 42
7
11
14
21
22
28
33
35
```

HINT:

> Make sure you understand the [odd_even.s](#) and [print_10.s](#) lecture examples.
>
> Start by choosing which register you will use for each variable.

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

```
$ 1521 autotest seven_eleven
```

## Autotest Results

100% of 70 students who have autotested seven_eleven.s so far, passed all autotest tests.

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

```
$ give cs1521 lab04_seven_eleven seven_eleven.s
```

You must run `give` before **Friday 03 July 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.

---

CHALLENGE EXERCISE — INDIVIDUAL:

# MIPS Tetrahedra

In the files for this lab, you have been given `tetrahedral.s`, a MIPS assembler program that reads a number and prints 42:

```
$ 1521 spim -f tetrahedral.s
Loaded: /home/cs1521/share/spim/exceptions.s
Enter a number: 42
42
```

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

```c
// Read a number n and print the first n tetrahedral numbers
// https://en.wikipedia.org/wiki/Tetrahedral_number

#include <stdio.h>

int main(void) {
    int i, j, n, total, how_many;

    printf("Enter how many: ");
    scanf("%d", &how_many);

    n = 1;

    while (n <= how_many) {
        total = 0;
        j = 1;

        while (j <= n) {
            i = 1;
            while (i <= j) {
                total = total + i;
                i = i + 1;
            }
            j = j + 1;
        }
        printf("%d\n", total);
        n = n + 1;
    }
    return 0;
}
```

For example:

```
$ 1521 spim -f tetrahedral.s
Loaded: /home/cs1521/share/spim/exceptions.s
Enter number: 5
1
4
10
20
35
$ 1521 spim -f tetrahedral.s
Loaded: /home/cs1521/share/spim/exceptions.s
Enter number: 12
1
4
10
20
35
56
84
120
165
220
286
364
```

> **HINT:**
>
> Make sure you understand the sum_100_squares.s lecture example.
>
> Started by choosing which register you will use for each variable.

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

```
$ 1521 autotest tetrahedral
```

## Autotest Results

97% of 33 students who have autotested tetrahedral.s so far, passed all autotest tests.

- 97% passed test *0 1 2 3*

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

```
$ give cs1521 lab04_tetrahedral tetrahedral.s
```

You must run `give` before **Friday 03 July 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.

---

CHALLENGE EXERCISE — INDIVIDUAL:
# Read & Execute MIPS Instructions

Write a MIPS assembler program `dynamic_load.s` which reads MIPS instructions as signed decimal integers until it reads the value -1, then executes the instructions.

**dynamic_load.s** should read instructions until it reads the value -1.

**dynamic_load.s** should then print a message, load the instructions, print another message, exactly as in the examples below.

For example, below is a tiny MIPS assembler program which prints 42. The comment on each line shows how the instruction is encoded, as a hexadecimal and as a signed decimal integer; it is this signed integer value that your program will read.

```
li $a0, 42  # 3404002a  -3422289878
li $v0, 1   # 34020001  -3422420991
syscall     # 0000000c          12
jr $ra      # 03e00008    65011720
```

This is what `dynamic_load.s` must do.

```
$ 1521 spim -f dynamic_load.s
Enter mips instructions as integers, -1 to finish:
-3422289878
-3422420991
12
65011720
-1
Starting executing instructions
42Finished executing instructions
```

The supplied files for the lab include files containing the instructions for some MIPS assembler programs from lectures. You can use these to test your program; for example:

```
$ cat add.instructions
-3422027759
-3421962215
19419168
663585
-3422420991
12
-3422289910
-3422420981
12
65011720
-1
$ 1521 spim -f dynamic_load.s <add.instructions
Loaded: /home/cs1521/share/spim/exceptions.s
Enter mips instructions as integers, -1 to finish:
Starting executing instructions
42
Finished executing instructions
$ 1521 spim -f dynamic_load.s <print10.instructions
Loaded: /home/cs1521/share/spim/exceptions.s
Enter mips instructions as integers, -1 to finish:
Starting executing instructions
1
2
3
4
5
6
7
8
9
10
Finished executing instructions
$ 1521 spim -f dynamic_load.s <sum_100_squares.instructions
Loaded: /home/cs1521/share/spim/exceptions.s
Enter mips instructions as integers, -1 to finish:
Starting executing instructions
338350
Finished executing instructions
```

If you want to experiment with your own tests, this command will give you any MIPS program as integers.

```
$ 1521 mips_instructions 42.s
-3422289878
-3422420991
12
65011720
-1
```

If you want to try creating your own test cases, here is some MIPS assembler that prints a message without using initialized data:

```
# print a string without using pre-initialized data
# for the dynamic load challenge exercise

main:
    li    $a0, 'H'        # printf("%c", 'Hi');
    li    $v0, 11
    syscall

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

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

    jr $ra
```

> **HINT:**
>
> This exercises involves writing only a small amount of code, but it is tricky to get right and will involve some thought and experimentation.

> **NOTE:**
>
> The input to `dynamic_load.s` will be between 1 and 1024 signed decimal integers: spim can't read hexadecimal.
>
> You can assume the instructions don't use initialized data. This mean printing strings in the normal way won't work.
>
> You can assume the instructions read in execute a `jr $ra` instruction to terminate.
>
> You can assume the instructions don't use jump instructions, except for `jr $ra`.

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

```
$ 1521 autotest dynamic_load
```

## Autotest Results

50% of 2 students who have autotested dynamic_load.s so far, passed all autotest tests.

- 50% passed test *42 add hi print10 sum_100_squares*

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

```
$ give cs1521 lab04_dynamic_load dynamic_load.s
```

You must run `give` before **Friday 03 July 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 **Friday 03 July 21:00** 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
```

**COMP1521 20T2: Computer Systems Fundamentals** is brought to you by
the [School of Computer Science and Engineering](#)
at the [University of New South Wales](#), Sydney.
For all enquiries, please email the class account at [cs1521@cse.unsw.edu.au](mailto:cs1521@cse.unsw.edu.au)
CRICOS Provider 00098G

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
```