# Laboratory Exercise 2
# Basic Arithmetic Operations

## Goals

After this laboratory exercise, you should understand polling, unsigned and signed integer arithmetic, have some understanding of IEEE standard floating point numbers, and know how a computer can manipulate text strings.

## Literature

- Patterson and Hennessy: Chapter 3.1–3.3, 3.6, Appendix A.10, or
  Brorsson: Chapter 3.1–3.3, 3.5-3.10, 4 Appendix A.
- MIPS Lab Environment Reference

## Preparation

Read the literature and this laboratory exercise in detail, and solve the home assignments.

### Home Assignment 1

Study the following simple assembly program, which is designed for investigation of arithmetic operations:

```
        # Laboratory Exercise 2, Home Assignment 1
        # Written by Jan Eric Larsson, 27 October 1998

#include <iregdef.h>

        .set noreorder
        .text
        .globl start
        .ent start

start:  jal     wait        # Wait for button click
        nop
        lui     s0, 0xbf90  # Load switch port address
        lb      s1, 0x0(s0) # Read first number from switches
        nop
        jal     wait        # Wait for button click
        nop
        lb      s2, 0x0(s0) # Read second number from switches
        nop
        addu    s3, s1, s2  # Perform an arithmetic operation
        sb      s3, 0x0(s0) # Write the result to LEDs
        b       start       # Repeat all over again
        nop

        ### Add code for wait subroutine here! ###

        .end start
```

Write a subroutine, `wait`, that will wait until the button K2 is pressed, then wait until the button is released, and finally, return. It should read from the address `0xbfa00000` until it receives a `1` in bit position 0 from this read operation. Next, it should read until the bit becomes zero again, and then return. This method of receiving

information from the outer world is called *polling*. The subroutine should be called with the instruction `jal wait`, as shown in the main program above.

### Home Assignment 2

Write an assembly program that reads a character from the keyboard and writes it to the console screen, over and over again. Any lowercase letter should be translated to the corresponding uppercase letter, while other characters should remain unchanged. You can use the *getchar* and *putchar* subroutines as described in the *MIPS Lab Environment Reference*.

## Simple Arithmetic and Polling

In this laboratory exercise, you will study how the computer can perform simple arithmetic operations, and see how it can read data from the real world by polling. This means that the computer repeatedly checks an input port to see if any data has appeared.

### Assignment 1

Start the MipsIt environment, create a project, and type in the program of Home Assignment 1 in a file. Also type in the `wait` subroutine that you wrote for Home Assignment 1. Build the program and upload it to the computer. Use the disassembler to verify that the code is correct.

### Assignment 2

Run the program and use the switches and LEDs on the lab computer board to investigate integer addition, as defined by the instruction `addu`. This operation performs integer addition for positive numbers. Try the additions 1 + 1, `0xFF` + 1, and others. Also investigate 32-bit additions. You may feed these longer values directly into the registers or into the data of a program. Try to load a 32-bit value from the switches by using the instruction `lb` and loading the value `0xFF`. What happens with the data in the register? Why? When do special or unusual results appear in addition?

### Assignment 3

Investigate the instructions `add`, `subu`, and `sub`, in a similar way, by replacing the `addu` instruction in the program above. For each instruction, explain how it works and why, and when its special cases appear. To investigate *overflow*, enter large numbers in the registers.

### Assignment 4

Investigate the instructions `multu` and `divu`, in the same way as in Assignment 2. Note that these instructions deliver their results in a special register, and that the assembler uses the instruction `mflo` to move the result to a standard register, as in the example below:

```
mult    t3, t4   # Multiply contents of t3 and t4
mflo    t5       # Move the result to t5
```

Why do `mult` and `div` use special result registers?

### Assignment 5

The program of Home Assignment 1 reads an 8-bit number from the switches but stores a 32-bit number in the register, using the instruction `lb`. What values are stored in the upper 24 bits? What happens if the instruction `lb` is replaced by `lbu`?

### Assignment 6

Investigate the effects of using `lb` and `lbu` and then answer the following questions. What is *sign extension* and what is it good for? Why are the conditions for overflow different in signed and unsigned arithmetic?

## Floating Point Numbers

The most common way of representing floating point numbers is to use the IEEE standard, described in the literature.

### Assignment 7

Study the definition of the IEEE standard in the course literature. Then calculate the bit pattern that represents the number 0.75 in single precision. Translate this bit pattern to a hexadecimal number and enter it in the simulator memory. Then use the disassembler to interpret the memory contents as floating point numbers, and verify that you translated the number correctly.

## Text Manipulation

An important task for computers is to handle text. Characters are usually coded so that one character corresponds to one byte, according to the ASCII table. You will now investigate two simple text handling programs.

### Assignment 8

Run the program of Home Assignment 2 and investigate how the program can echo key strokes, in other words, how it can read key strokes and type the corresponding characters on the text terminal.

### Assignment 9

The following C program calls the standard subroutine *printf* to write a text string to the console.

```
/*
 *  Laboratory Exercise 2, Assignment 9
 *  Written by Kernighan and Ritchie, 1978
 *
 */

main()
{
  printf("hello, world\n");
}
```

Create a C minimal project and save this program on a file. Build and upload it to the lab computer, and execute it to see what happens. Then test the effects of the special characters \n, \t, \b, \r, \", and \\. Find their numerical values and their place in the ASCII table. Finally, study the ASCII table and note where different types of characters are located.

## Conclusions

Before you finish the laboratory exercise, think about the questions below:

- What is polling?
- How does the computer know whether a number is signed or unsigned?
- What is an overflow?
- What happens when an overflow occurs?
- Why are not all numbers represented by floating point?
- What is double precision?
- Why are there non-printable characters?