ECE 414 – Lab 4
Section 2.4 – Performance/Size Measurement
Hayden Dodge & Connor Wyniarcyzk

2.4 Performance/Size Measurement

Time (us) per operation vs. type and operation

	+	-	*	/
uint8_t	0.216500	0.216500	0.266500	0.517500
uint16_t	0.216500	0.216500	0.267000	0.717000
uint32_t	0.216500	0.216500	0.292000	1.093000
uint64_t	0.442000	0.442000	0.968000	3.196500 *
float	1.569000	1.744500	1.469000	3.597000
double	1.469000	1.644000	1.469000	3,546000

Note [1]: 0.717000 us when using the original test case with integer division result of 0.

Size (bytes) per operation vs. type and operation

	+	-	*	/
uint8_t	20.8	20.8	24.8	32.8
uint16_t	20.8	20.8	24.8	32.8
uint32_t	21.2	21.2	25.2	33.2
uint64_t	51.6	51.6	83.6	56
float	25.6	25.6	25.6	25.6
double	25.6	25.6	25.6	25.6

We can see that for numeric datatypes that division takes the longest amount time to perform. If the datatype is an integer, multiplication takes more time that addition and subtraction but less time than division. For floating point datatypes, addition, subtraction, and multiplication take about the same amount of time. When comparing the size of integer datatypes, size only matters when it exceeds the bit capacity of the processor. Since we are using a 32-bit processor for these tests there is not significant increase in operating time until the 64-bit integer is used. Further, the is an increase in operating time for all operation when using a floating point datatype over an integer. There is not a significant difference in operating time between float and double precision floating point types.

In the size table, it is seen that the size of addition and subtraction is the same for all datatypes. Furthermore, there is a slight increase in function size as the size of the integer datatype increases. Division is one the more complicated algorithms for integers and thus has greater complexity than multiplication and addition/subtraction. It is interesting that the division function isa smaller in size than the multiplication function for uint64_t. The multiplication and division of floating points is about the same complexity as addition and subtraction as those functions are about the same size.

We had trouble with the clock for the UART functionality. We forgot to add the PRAGMA statements needed to setup to peripheral clock f_{PB}. We did battle once compiler and linker when using header files and the static keyword to create “Namespaces”. This we won a narrow margin. The next will surely go more swiftly. We had trouble finding the map file needed to complete the performance section of the lab.

Code Listings - SECTION 2.1 – 2.2:
main.c

```
#include <stdlib.h>
#include <p32xxxx.h>
#include "Utilities/uart.h"
// #include "Utilities/uart.c"
#include <plib.h>

// Configure clocks
#pragma config FNOSC = FRCPLL, POSCMOD = OFF
#pragma config FPLLIDIV = DIV_2, FPLLMUL = MUL_20 //40 MHz
#pragma config FPBDIV = DIV_1, FPLLODIV = DIV_2 // PB 40 MHz
#pragma config FWDTEN = OFF,  FSOSCEN = OFF, JTAGEN = OFF

void test_blocking() {
	while(1){
		char input = UART.busy_read();
		UART.busy_write(input);

		UART.busy_write('\r');
		UART.busy_write('\n');
	}
}

void test_non_blocking(){
	while(1){
		while(!UART.read_ready()){}
		char input = UART.nb_read();

		while(!UART.write_ready()){}
		UART.nb_write(input);
	}
}

void main() {
	UART.init();

	// test_blocking();
	test_non_blocking();
}
```

Code Listings - SECTION 2.1 – 2.2:
uart.h

```c
/**      UART
 * Authors: Hayden Dodge, Connor Winiarczyk
 *
 * this module provides functions for interacting with the build in
 * PIC UART hardware. Methods exist for reading and writing single characters
 * in both blocking (busy_xxx()) and non-blocking (nb_xxx()) varieties.
 *
 * The simplest way to interact with this module is by calling UART.write_string()
 *       With the desired value to be printed
 *
 *    NOTE: it is important to call UART.init() in your programs main method before
 *                trying to interact with this module
 */

#ifndef UART_H
#define UART_H

#include <p32xxxx.h>
#include <inttypes.h>
#include <plib.h>

//Defines an interface for talking with the UART module
typedef struct uart_interface {

      void (*init)(); // void UART.init()

      void (*busy_write)(char);     // void UART.busy_write(char)
      char (*busy_read)();          // char UART.busy_read()

      void (*nb_write)(char); // void UART.nb_write(char)
      char (*nb_read)();            // char UART.nb_read()

      int (*write_ready)();   // int UART.write_ready()
      int (*read_ready)();    // int UART.read_ready()

      void (*write_string)(char[]); // int UART.write_string(char[])
} uart_interface;

// instance of the interface we will be using
uart_interface UART;

#endif
```

Code Listings - SECTION 2.1 – 2.2:
uart.c

```
#include "uart.h"

void uart_test(){

}

static void init(){
	TRISA = ~0;
	ANSELA = 0;

	U1RXR = 0;
	RPA0R = 1;
	U1BRG = 259;

	U1MODE &= ~(1 << 3);
	U1STA |= ((1 << 12) | (1 << 10));
	U1MODE |= (1 << 15);
	}

	static void busy_write(char input) {
		while(1) {
			int isReady = !(U1STA & (1 << 9));

			if(isReady) {
				U1TXREG = input;
				break;
			}
		}
	}

	static char busy_read(){
		while(1) {
			int dataReady = (U1STA & (1 << 0));

			if(dataReady) {
				return U1RXREG;
			}
		}
	}

	static int write_ready(){
		return !(U1STA & (1 << 9));
	}

	static int read_ready(){
		return (U1STA & (1 << 0));
	}

	static void nb_write(char input) {
		if(U1STA & (1 << 9)) return;
		U1TXREG = input;
	}

	static char nb_read() {
		if(U1STA & (1 << 0)){
			return U1RXREG;
```

```
        } else {
            return '\0';
        }
    }

    static void write_string(char input[]){
        int index = 0;
        while(input[index] != '\0'){
            busy_write(input[index]);
            index ++;
        }
    }

    uart_interface UART = {
        &init,

        &busy_write,
        &busy_read,

        &nb_write,
        &nb_read,

        &write_ready,
        &read_ready,

        &write_string
    };
```

Code Listings - SECTION 2.3:

main.c

```
/*
 * ECE 414
 * File:   main.c
 * Author: dodgeh
 *
 * For lab 04 section 2.3 Interrupt Timer
 * Template from Lab04 description.
 *
 */

#include <p32xxxx.h>
#include <inttypes.h>
#include <plib.h>
#include "Utilities/timer_ms/timer_ms.h"
#include "Utilities/uart.h"

#pragma config FNOSC = FRCPLL, POSCMOD = OFF
#pragma config FPLLIDIV = DIV_2, FPLLMUL = MUL_20 //40 MHz
#pragma config FPBDIV = DIV_1, FPLLODIV = DIV_2 // PB 40 MHz
#pragma config FWDTEN = OFF,  FSOSCEN = OFF, JTAGEN = OFF

#define NUM_ITERATIONS 100000
#define NUM_REPS 10

void test_uint8_mult(){
    uint32_t i;
    uint8_t i1, i2, i3;
}

main(){
    TIMER.init();
    UART.init();

    // UART.write_string("test");

    // Turn on system interrupts
    INTEnableSystemSingleVectoredInt();

    while(1) {
        while(1) {
            int isReady = !(U1STA & (1 << 9));

            if(isReady) {
                U1TXREG = 'c';
                break;
            }
        }
    }

    // Show that the interrupt is working!
    TRISACLR = 0x01; // Set RA0 to be an output
    while (1){
        TIMER.delay_ms(1000);
        // Turn LED on
        PORTASET = 0x1;
        TIMER.delay_ms(1000);
        // Turn LED off
        PORTACLR = 0x1;
    }
}
```

Code Listings - SECTION 2.3:

timer_ms.h

```
#ifndef TIMER_MS_H
#define TIMER_MS_H

#include <p32xxxx.h>
#include <inttypes.h>
#include <plib.h>

typedef struct timer_interface{
      void (*init)();
      int (*read)();
      void (*delay_ms)(int);
} timer_interface;

timer_interface TIMER;

#endif
```

Code Listings - SECTION 2.3:

timer_ms.c

```
#include "timer_ms.h"

static volatile uint32_t timer_ms_count;

void __ISR( 0, ipl1auto) InterruptHandler (void){
    timer_ms_count++;
    // !!! Put code here to clear the interrupt flag
    mT1ClearIntFlag();
}

static void init(){
      // !!! 0. Put code to set up Timer 1 like you did before.
    // !!! Use a prescaler of 1 or 8 this time. Set PR1 to reset every 1 ms.
    T1CON = 0x8000; // TMR1 on, prescale 1:1
    PR1 = 40000; // 1 ms timer rst, T_delay = Prescaler * DELAY / Clock_Freq

    // Initialize interrupts
    // !!! 1. Set T1 interrupt source to have a priority of 1
    mT1SetIntPriority(1);

    // !!! 3. Enable the T1 interrupt source
   mT1IntEnable(1);
}

static int read(){
      return timer_ms_count;
}

static void delay_ms(int time){
      uint32_t last_count;
      while(timer_ms_count < last_count + time){

      }
}

timer_interface TIMER = {
      &init,
      &read,
      &delay_ms
};
```

Code Listings - SECTION 2.4:

main.c

```
/*
 * File:   main.c
 * Author: dodgeh
 *
 * Created on September 27, 2018, 1:42 AM
 */

#include <p32xxxx.h>
#include <inttypes.h>
#include <stdio.h> // for sprintf

#include "Utilities/timer_ms/timer_ms.h"
#include "Utilities/uart.h"

#pragma config FNOSC = FRCPLL, POSCMOD = OFF
#pragma config FPLLIDIV = DIV_2, FPLLMUL = MUL_20 //40 MHz
#pragma config FPBDIV = DIV_1, FPLLODIV = DIV_2 // PB 40 MHz
#pragma config FWDTEN = OFF,  FSOSCEN = OFF, JTAGEN = OFF

// Number of iterations for testing. You may need to play with this number.
// If it is too short, you may not get a very accurate measure of performance.
// Too long and you will have to wait forever.
#define NUM_ITERATIONS 100000

// This is the number of times you repeat the operation within the loop.
// You want to repeat enough times that the loop overhead is small for
// the simplest operations.
#define NUM_REPS 20

#define NUM_TESTCASES 24

uint8_t buffer[64];

void test_uint8_add(){

    uint32_t i;
    uint8_t i1, i2, i3;
    i1 = 15;
    i2 = 26;
    for (i=0; i<NUM_ITERATIONS; i++){
        // Make sure NUM_REPS is the same as the number
        // of repeated lines here.
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;

        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
```

```c
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
    }
}

void test_uint8_sub(){

    uint32_t i;
    uint8_t i1, i2, i3;
    i1 = 15;
    i2 = 26;
    for (i=0; i<NUM_ITERATIONS; i++){
        // Make sure NUM_REPS is the same as the number
        // of repeated lines here.
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;

        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
    }
}

void test_uint8_mult(){

    uint32_t i;
    uint8_t i1, i2, i3;
    i1 = 15;
    i2 = 26;
    for (i=0; i<NUM_ITERATIONS; i++){
        // Make sure NUM_REPS is the same as the number
        // of repeated lines here.
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;

        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
    }
}

void test_uint8_div(){

    uint32_t i;
    uint8_t i1, i2, i3;
    i1 = 15;
    i2 = 26;
    for (i=0; i<NUM_ITERATIONS; i++){
        // Make sure NUM_REPS is the same as the number
        // of repeated lines here.
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;

        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
    }
}

void test_unint16_add(){

    uint32_t i;
    uint16_t i1, i2, i3;
    i1 = 12734;
    i2 = 89139;
    for (i=0; i<NUM_ITERATIONS; i++){
        // Make sure NUM_REPS is the same as the number
        // of repeated lines here.
```

```c
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;

        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
    }
}

void test_unint16_sub(){

    uint32_t i;
    uint16_t i1, i2, i3;
    i1 = 12734;
    i2 = 89139;
    for (i=0; i<NUM_ITERATIONS; i++){
        // Make sure NUM_REPS is the same as the number
        // of repeated lines here.
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;

        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
    }
}

void test_unint16_mult(){
    uint32_t i;
    uint16_t i1, i2, i3;
    i1 = 12734;
    i2 = 89139;
    for (i=0; i<NUM_ITERATIONS; i++){
        // Make sure NUM_REPS is the same as the number
        // of repeated lines here.
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;

        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
    }
}

void test_unint16_div(){

    uint32_t i;
    uint16_t i1, i2, i3;
    i1 = 12734;
    i2 = 89139;
    for (i=0; i<NUM_ITERATIONS; i++){
        // Make sure NUM_REPS is the same as the number
        // of repeated lines here.
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;

        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
    }
}

void test_unint32_add(){

    uint32_t i;
    uint32_t i1, i2, i3;
    i1 = 294967296;
    i2 = 89496789;
    for (i=0; i<NUM_ITERATIONS; i++){
        // Make sure NUM_REPS is the same as the number
        // of repeated lines here.
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;

        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
    }
}

void test_unint32_sub(){

    uint32_t i;
    uint32_t i1, i2, i3;
    i1 = 294967296;
    i2 = 89496789;
    for (i=0; i<NUM_ITERATIONS; i++){
        // Make sure NUM_REPS is the same as the number
        // of repeated lines here.
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;

        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
    }
}

void test_unint32_mult(){

    uint32_t i;
    uint32_t i1, i2, i3;
    i1 = 294967296;
    i2 = 89496789;
    for (i=0; i<NUM_ITERATIONS; i++){
        // Make sure NUM_REPS is the same as the number
        // of repeated lines here.
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;

        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
    }
}

void test_unint32_div(){

    uint32_t i;
    uint32_t i1, i2, i3;
    i1 = 294967296;
    i2 = 89496789;
    for (i=0; i<NUM_ITERATIONS; i++){
        // Make sure NUM_REPS is the same as the number
        // of repeated lines here.
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;

        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
    }
}

void test_unint64_add(){

    uint32_t i;
    uint64_t i1, i2, i3;
    i1 = 1273489139;
    i2 = 8913912734;
    for (i=0; i<NUM_ITERATIONS; i++){
        // Make sure NUM_REPS is the same as the number
        // of repeated lines here.
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;

        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
    }
}

void test_unint64_sub(){

    uint32_t i;
    uint64_t i1, i2, i3;
    i1 = 1273489139;
    i2 = 8913912734;
    for (i=0; i<NUM_ITERATIONS; i++){
        // Make sure NUM_REPS is the same as the number
        // of repeated lines here.
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;

        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
    }
}

void test_unint64_mult(){

    uint32_t i;
    uint64_t i1, i2, i3;
    i1 = 1273489139;
    i2 = 8913912734;
    for (i=0; i<NUM_ITERATIONS; i++){
        // Make sure NUM_REPS is the same as the number
        // of repeated lines here.
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;

        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
    }
}
```

```
void test_unint64_div(){

    uint32_t i;
    uint64_t i1, i2, i3;
    i1 = 8913912734;
    i2 = 1273489139;
    for (i=0; i<NUM_ITERATIONS; i++){
        // Make sure NUM_REPS is the same as the number
        // of repeated lines here.
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;

        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
    }
}

void test_float_add(){

    uint32_t i;
    float i1, i2, i3;
    i1 = 12.734;
    i2 = 8913.9;
    for (i=0; i<NUM_ITERATIONS; i++){
        // Make sure NUM_REPS is the same as the number
        // of repeated lines here.
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;

        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
    }
}

void test_float_sub(){

    uint32_t i;
    float i1, i2, i3;
    i1 = 12.734;
    i2 = 8913.9;
    for (i=0; i<NUM_ITERATIONS; i++){
        // Make sure NUM_REPS is the same as the number
        // of repeated lines here.
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;

        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
    }
}

void test_float_mult(){

    uint32_t i;
    float i1, i2, i3;
    i1 = 12.734;
    i2 = 8913.9;
    for (i=0; i<NUM_ITERATIONS; i++){
        // Make sure NUM_REPS is the same as the number
        // of repeated lines here.
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
    }
}

void test_float_div(){

    uint32_t i;
    float i1, i2, i3;
    i1 = 12.734;
    i2 = 8913.9;
    for (i=0; i<NUM_ITERATIONS; i++){
        // Make sure NUM_REPS is the same as the number
        // of repeated lines here.
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;

        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
    }
}

void test_double_add(){

    uint32_t i;
    double i1, i2, i3;
    i1 = 12.73491746298;
    i2 = 99999999978949878913.9;
    for (i=0; i<NUM_ITERATIONS; i++){
        // Make sure NUM_REPS is the same as the number
        // of repeated lines here.
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;

        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
        i3 = i1+i2;
    }
}

void test_double_sub(){

    uint32_t i;
    double i1, i2, i3;
    i1 = 12.73491746298;
    i2 = 99999999978949878913.9;
    for (i=0; i<NUM_ITERATIONS; i++){
        // Make sure NUM_REPS is the same as the number
        // of repeated lines here.
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;

        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
        i3 = i1-i2;
    }
}

void test_double_mult(){

    uint32_t i;
    double i1, i2, i3;
    i1 = 12.73491746298;
    i2 = 99999999978949878913.9;
    for (i=0; i<NUM_ITERATIONS; i++){
        // Make sure NUM_REPS is the same as the number
        // of repeated lines here.
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;

        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
        i3 = i1*i2;
    }
}

void test_double_div(){

    uint32_t i;
    double i1, i2, i3;
    i1 = 12.73491746298;
    i2 = 99999999978949878913.9;
    for (i=0; i<NUM_ITERATIONS; i++){
        // Make sure NUM_REPS is the same as the number
        // of repeated lines here.
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;

        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
        i3 = i1/i2;
    }
```

```
}


void (*testcases[])() = {
    &test_uint8_add,
    &test_uint8_sub,
    &test_uint8_mult,
    &test_uint8_div,
    &test_unint16_add,
    &test_unint16_sub,
    &test_unint16_mult,
    &test_unint16_div,
    &test_unint32_add,
    &test_unint32_sub,
    &test_unint32_mult,
    &test_unint32_div,
    &test_unint64_add,
    &test_unint64_sub,
    &test_unint64_mult,
    &test_unint64_div,
    &test_float_add,
    &test_float_sub,
    &test_float_mult,
    &test_float_div,
    &test_double_add,
    &test_double_sub,
    &test_double_mult,
    &test_double_div
};

void main(){
    UART.init();
    TIMER.init();
    INTEnableSystemSingleVectoredInt();

    uint32_t t1, t2;

    int i;
    for(i = 0; i < NUM_TESTCASES; i++){
        UART.write_string("Performance Summary: Time per operation
statistics\r\n");
        // Test multiplying bytes
        t1 = TIMER.read();
        testcases[i]();
        t2 = TIMER.read();

        // Print timing result. Doubles OK here. Not time or space critical code.
        sprintf(buffer, " : %.06f us per operation\r\n",
        (double)(t2-t1)/(double)NUM_ITERATIONS/(double)NUM_REPS*1000.0);
        UART.write_string(buffer);

        if(i % 4 == 3){
            UART.busy_write('\r');
            UART.busy_write('\n');
        }
    }
    while (1); // When done, wait forever.
}
```