Homework 4

Qingyuan Chen

Chapter 3, problems 3 (for part a, ports A and B, and what pin is B0), 4,5,7,9 (for TRISB instead of TIRSD).

Chapter 4, problems 1, 2

Write a library for the ultrasonic range finder (HC-SR04) using the template sr04.c and sr04.h in the Files section of Canvas. Build the HC-SR04 circuit and test the sensor. Turn in your code and a video demonstrating the PIC printing the distance from the sensor in meters 4 times per second.

Chapter 3

3

Refer to the Memory Organization section of the Data Sheet and Figure 2.1.

a.

Referring to the Data Sheet, indicate which bits, 0-31, can be used as input/outputs for each of Ports B through G. For the PIC32MX795F512H in Figure 2.1, indicate which pin corresponds to bit 0 of port E (this is referred to as RE0).

The input and output of each port is with `PORTx` registers.

For ports A and B:

`PORTA` register bits <10:7> , <4:0> are for PORTA10:PORTA4

`PORTB` register bits <15:0> are used for PORTB15:PORT0

For `PIC32MX170F256B` port B0 is at pin 4.

b.

The SFR INTCON refers to “interrupt control.” Which bits, 0-31, of this SFR are unimplemented? Of the bits that are implemented, give the numbers of the bits and their names.

For `INTCON` , bit <31:13> <11> <7:5> are unimplemented.

bit 12 is MVEC bit <10:8> is TPC bit <4:0> is INT4EP to INT0EP

4

Modify simplePIC.c so that both lights are on or off at the same time, instead of opposite each other. Turn in only the code that changed.

The only line needs changing is:

```
LATBbits.LATB4 = 1;
```

This basically set the pins to the same state on start, so the toggling action after this will be in sync.

5

Modify simplePIC.c so that the function delay takes an int cycles as an argument. The for loop in delay executes cycles times, not a fixed value of 1,000,000. Then modify main so that the first time it calls delay, it passes a value equal to MAXCYCLES. The next time it calls delay with a value decreased by DELTACYCLES, and so on, until the value is less than zero, at which time it resets the value to MAXCYCLES. Use #define to define the constants MAXCYCLES as 1,000,000 and DELTACYCLES as 100,000. Turn in your code.

See `simplePIC_param_delay.c`.

7

The processor.o file linked with your simplePIC project is much larger than your final .hex file. Explain how that is possible.

The .o file is a raw machine code binary. Specifically the `processor.o` includes all the SFR virtual addresses, which is lots of information.

The .hex file is a compressed version of the code. Which has skipped all the memory address that are not being used. Also it is a compressed format by itself.

9

Give three C commands, using TRISDSET, TRISDCLR, and TRISDINV, that set bits 2 and 3 of TRISD to 1, clear bits 1 and 5, and flip bits 0 and 4.

for TRISB instead of TIRSD

```
// Set TRISD bit 2 and 3 to 1
TRISBSET = 0x0C; // which is 0b 0000 1100
// clear TRISD bit 1 and 5;
TRISBCLR = 0x33; // which is 0b 0010 0010
// flip bit 0 and 4
TRISBINV = 0x // which is 0b 0001 0001
```

Chapter 4

1

Identify which functions, constants, and global variables in NU32.c are private to NU32.c and which are meant to be used in other C files.

assuming `NU32.c` and `NU32.h` are in the `embedded_computing_code/01_Quickstart_nu32dip/skeleton` folder.

Since the convention is to have definition in header file, declaration in source file, any function or variables in `NU32.c` that is not first declared in header will be "private" to `NU32.c`.

`NU32.h` contains the declaration for

```
#define NU32_LED1 LATFbits.LATF0    // LED1 on the NU32 board
#define NU32_LED2 LATFbits.LATF1    // LED2 on the NU32 board
#define NU32_USER PORTDbits.RD7     // USER button on the NU32 board
#define NU32_SYS_FREQ 80000000ul    // 80 million Hz

void NU32_Startup(void);
void NU32_ReadUART3(char * string, int maxLength);
void NU32_WriteUART3(const char * string);
```

So these functions: `NU32_Startup` `NU32_ReadUART3` `NU32_WriteUART3` are provided to others

These macro values are defined in header for others to use: `NU32_LED1` `NU32_LED2` `NU32_USER` `NU32_SYS_FREQ`

```
#define NU32_DESIRED_BAUD 230400
```

are private to the source file only.

And there are no global or const variable.

2

You will create your own libraries.

a.

Remove the comments from invest.c in Appendix A. Now modify it to work on the NU32 using the NU32 library. You will need to replace all instances of printf and scanf with appropriate combinations of sprintf, sscanf, NU32_ReadUART3 and NU32_WriteUART3. Verify that you can provide data to the PIC32 with your keyboard and display the results on your computer screen. Turn in your code for all the files, with comments where you altered the input and output statements.

see `invest.c`

b.

Split invest.c into two C files, main.c and helper.c, and one header file, helper.h. helper.c contains all functions other than main. Which constants, function prototypes, data type definitions, etc., should go in each file? Build your project and verify that it works. For the safety of future helper library users, put an

include guard in helper.h. Turn in your code and a separate paragraph justifying your choice for where to put the various definitions.

see `helper.c` amd `helper.h` for detail

The header needs to declear all the public facing function, aka function prototypes, which is:

```
int getUserInput(Investment *invp);
void calculateGrowth(Investment *invp);
void sendOutput(double *arr, int years);
```

In addition, the public data structure and macro constant also needs to be in header so others can access it.

```
#define MAX_YEARS 100
typedef struct {
  double inv0;
  double growth;
  int years;
  double invarray[MAX_YEARS + 1];
} Investment;
```

everything else should stay in source.

C.

Break invest.c into three files: main.c, io.c, and calculate.c. Any function which handles input or output should be in io.c. Think about which prototypes, data types, etc., are needed for each C file and come up with a good choice of a set of header files and how to include them. Again, for safety, use include guards in your header files. Verify that your code works. Turn in your code and a separate paragraph justifying your choice of header files.

The headers are split in three.

As a common convention, the `io.c` and `claculate.c` each get a header `io.h` and `calculate.h`. This two header will host their function declaration (prototypes).

io.h have

```
int getUserInput(Investment *invp);
void sendOutput(double *arr, int years);
```

claculate.h have

```
void calculateGrowth(Investment *invp);
```

Since both file depends on the `Investment` datatype. This is moved to its own header `invest_type.h`, which have

```
#define MAX_YEARS 100 // constant indicating max number of years to track

typedef struct {
  double inv0;                    // initial investment
  double growth;                  // growth rate, where 1.0 = zero growth
  int years;                      // number of years to track
  double invarray[MAX_YEARS + 1]; // investment array   ==SecA.4.9==
} Investment;                     // the new data type is called Investment
```

This way, each compilation unit only includes the minimal needed content from the headers.