**INTERRUPTS**

**Aayahna M. Herbert**

**October 10, 2017**

**ECE 3720 Section 007**

**Microcontroller Interfacing Laboratory**

**Abstract:** A circuit was built in order to send an interrupt to the circuit board’s LED lights. The ISR was to display the value fifteen on the four LEDs, wait a few seconds, and then leave the interrupt and return normal functionality as if nothing happened.

**INTRODUCTION**

The purpose of this lab was to design a circuit that sends an input to the microcontroller from the debouncer output. The microcontroller would then output signals to four LEDs that represent binary numbers 0 through 15 when turned on. When done correctly, as time passes, the LEDs will increment from 0 to 15 in binary; when the interrupt button is pushed, all of the lights should light up for a brief moment, then the lights will go back to their original function.

**EXPERIMENTAL PROCEDURES**

The equipment used include one PIC32 MC, NI-ELVIS II board, a 74LS00 NAND gate chip, and an interrupt button. One thing to note is that the code for this lab uses the exact code from Lab 1 as a base to build off of. The first thing to be done, to avoid forgetting it later, is to set ports B0 through B3 as outputs and B7 as an input. Pins B0 through B3 are used to enable the board’s LED lights. Because we are dealing with one interrupt signal, the register INT0 should be enabled, set in to high direction, be interrupted on a high value, and have a priority of one by using INTCON, IECx, IFSx, and IPCx3. In addition to the interrupt, it must also be globally enabled by adding INTEnableSystemMultiVectoredInt() to the main function. With the main function comes the ISR function for the specific register number in use, void \_\_ISR(3) interruption (void). Inside of this function goes the commands for what should happen to the LED lights when the interrupt button is pressed. Because my interrupt will have all of the lights come on simultaneously, the function will have Ports B0 through B3 set to one using LATX; to indicate the end of the interrupt, the flag must be reset (set to zero) using IFSx followed by calling the delay function made from Lab 1.

**RESULTS**

After the circuit was hooked up and ready to be tested, the program detected no compiling errors when ran so the board could then be tested. While running tests, when the interrupt button was pressed, the lights all lit up simultaneously for few seconds and then went back to their previous actions of incrementing from 0 to 15 in binary.

**DISCUSSION**

Problems arose initially when the LEDs didn’t light all up at once when the interrupt button was pressed. The LEDs and interrupt button were tested individually to make sure none of the ones in use were broken. The code and wiring were second-checked by the TA and the wires all came from a new pack recently purchased to avoid the possibility of dead wires being used and affecting the lab results. The source of the error was the missing global enabler function that should have been included at the beginning of the main function; it was erased when copying over the code from Lab 1. When added back, the lab results came out to be successful.

**CONCLUSIONS**

In general, the take-away of this lab was to get hands-on experience with interrupts by learning what they are, how they are set up, and what the correct way to handle them is.

**FIGURES AND TABLES**

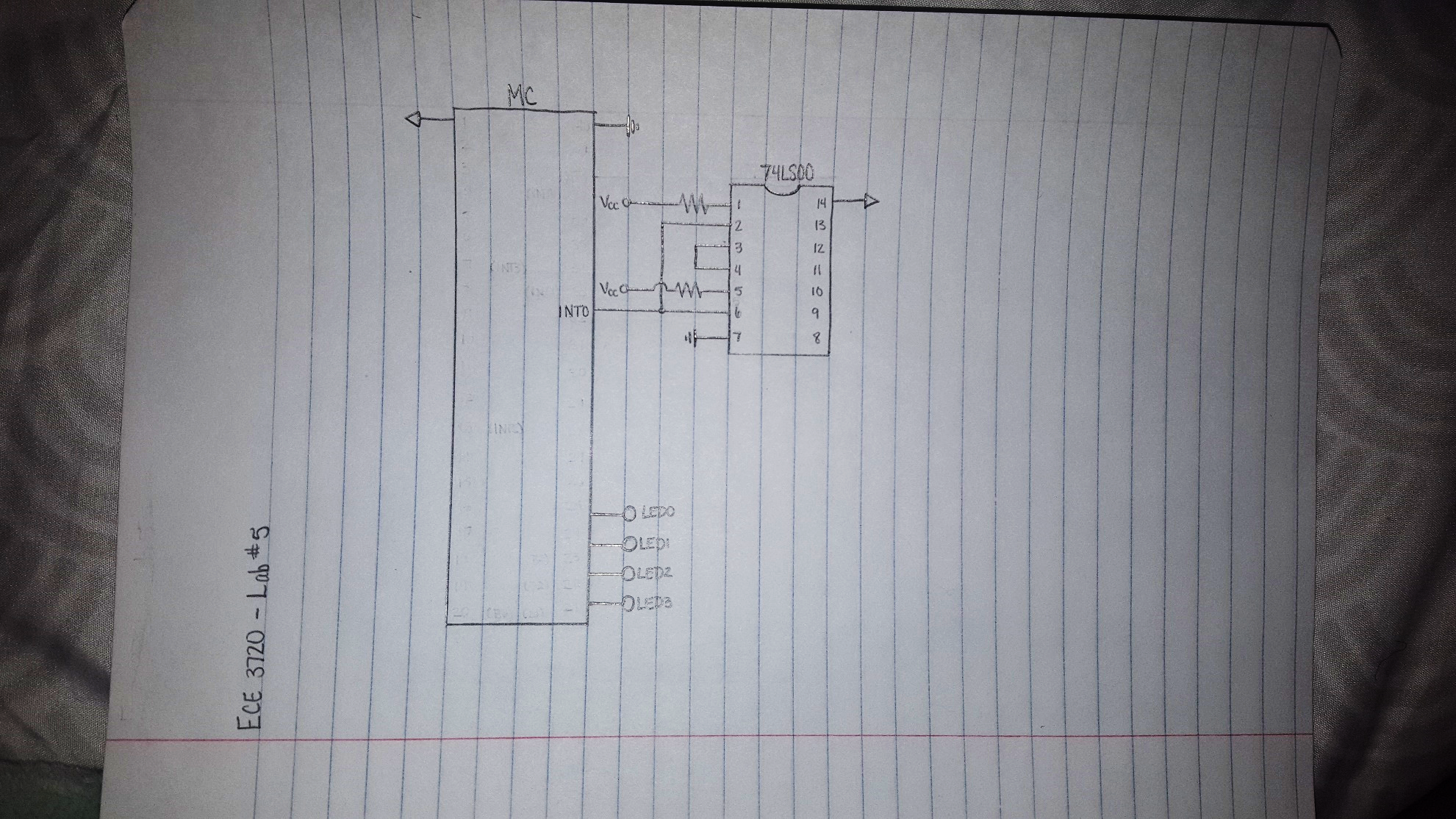


Figure 1: Circuit Schematic

**CODE**

#include<plib.h>

delay()

{

int i, j;

for(i = 0; i < 500; i++)

for(j = 0; j < 500; j++);

}

void \_\_ISR(3) **interruption**(void)

{

LATBbits.LATB0 = 1;

LATBbits.LATB1 = 1;

LATBbits.LATB2 = 1;

LATBbits.LATB3 = 1;

IFS0bits.INT0IF = 0;

delay();

}

main()

{

INTEnableSystemMultiVectoredInt();

int count = 0;

TRISBbits.TRISB0 = 0;

TRISBbits.TRISB1 = 0;

TRISBbits.TRISB2 = 0;

TRISBbits.TRISB3 = 0;

TRISBbits.TRISB7 = 1;

INTCONbits.INT0EP = 1;

IEC0bits.INT0IE = 1;

IFS0bits.INT0IF = 1;

IPC0bits.INT0IP = 1;

while(1)

{

LATB = count;

count++;

if(count < 15)

count = 0;

delay();

}

}