CS224

SECTION 03 / SECTION04

LAB07 PRELIMINARY REPORT

AYBÜKE CEREN DURAN/21302686

ROLA YOUNİS/21600537

b-)

**TRIS Registers:** they configure the data direction flow through port I/O pin(s). The TRIS register bits determine whether a PORT I/O pin is an input or an output.

* A TRIS bit set = 1 helps for configuring it as input
* A TRIS bit set = 0 helps for configuring it as output
* A read from TRIS helps for reading the last value written to TRIS register
* When reset is active, all devices are configured to input

**PORT Registers:** they allow I/O pins to be accessed (read). PORT provide inputs and outputs to access microcontroller.

* A write to PORT register is useful for writing to the corresponding LAT register. If any pins configured as output, they are corrected to input.
* A write to a PORT register corresponds to (same as) a write to a LAT register
* A read from a PORT register is useful for reading the value in the module (such as button matrix or leds) the I/O connection is connected to.

**LAT Registers:** LAT registers (PORT data latch) is responsible for holding data written to port I/O pin(s).

* Holds data written to the module pins are connected to.
* A write to a LAT register is responsible for transfering the data to the I/O device. Device set to output is updated to input.
* Read from a LAT register reads the data in LAT register, not from the I/O device.

**ODC Registers:** Each I/O pin can be individually configured for either normal digital output or open-drain output. The Open-Drain Control register, ODCx, associated with each I/O pin, controls this.

If the ODC bit for an I/O pin is ‘1’, that pin acts as an open-drain output. Else, it’s configured as a normal digital output. ODC bit is only valid for output pins. After a reset, all bits in ODCx register is set to ‘0’.

Open-drain feature allows generation of outputs higher than VDD on any desired digital only pins by using external-pull-up resistors. Maximum open-drain voltage allowed is the same as the maximum VIH specifications. ODC register settings takes effect in all I/O modes, allowing the output to behave as an open-drain even if a peripheral is controlling the pin. Although user could achieve the same result by manipulating the corresponding LAT and TRIS bits, this procedure will not allow the peripheral to operate in open-drain mode. Since I2C pins are already open-drain pins, ODCx settings don’t affect the I2C pins. Also, ODCx settings don’t affect the JTAG output characteristics as the JTAG scan cells are inserted between ODCx logic and I/O device.

**In part 2a, we used TRISx, and PORTx registers. In part 2b, we used TRISx, and PORTx registers.**

// Rule of thumb: Always read inputs from PORTx and write outputs to LATx.

// If you need to read what you set an output to, read LATx.

c-) C code for part2.a

int main(void) {

int leds;

TRISD = 0xFFFF; //port d takes input and reads the 2 pushbuttons

TRISA = 0x0000; // port a outputs the pattern of the leds and turns them on or off //accordingly

leds = 0x0088 // the 8-bit pattern 10001000 to the leds

Delay\_ms(1000); // delaying with 1.0 seconds

//RE7\_bit IS USED AS DIRECTION INPUT

while (1) {

// rotates to the right if DIR = 0 and rotates to the left if DIR = 1

PORTA = leds; //display on leds

if(RE7\_bit) { //rotate left and mask bits

if (RE6\_bit){ // if EN = 1 the pattern is displayed

leds<<1 & 0xFF;

leds<<7 & 0xFF;

Delay\_ms(1000); // delaying with 1.0 seconds

}

}

else { // rotate right and mask bits

//RE\_6 BİT IS USED ENABLE INPUT

if (RE6\_bit){ // if EN = 1 the pattern is displayed

leds>>1 & 0xFF;

leds >>7 & 0xFF;

Delay\_ms(1000); // delaying with 1.0 seconds

}

if (! RE6\_bit){ // if EN = 0 the pattern is frozen

Delay\_ms(1000); // delaying with 1.0 seconds

PORTA = 0xffff;

}

}

}

}

d-) C code for part2. B

int fourthDig,thirdDig,secondDig,firstDig; //digits to be displayed in 7seg

int xNumber; //our x value

int output; //our f(x)

int slowTemp; // temporary needed to make execution slower and visible

int dispNr; //which digit will be represented

int a,b;

int modulus(int number){ //compute the modulus of a number with 10

a = number / 10;

b = a \* 10;

return number - b; //modulus

}

void setNUM(int dispNr){ //helper function for the configurations of each digit

if (dispNr == 9) {

PORTD = 0b01101111;

}

else if (dispNr == 8){

PORTD = 0b01111111;

}

else if (dispNr == 7) {

PORTD = 0b00000111;

}

else if (dispNr == 6) {

PORTD = 0b01111101;

}

else if (dispNr == 5) {

PORTD = 0b01101101;

}

else if (dispNr == 4) {

PORTD = 0b01100110;

}

else if (dispNr == 3) {

PORTD = 0b01001111;

}

else if (dispNr == 2) {

PORTD = 0b01011011;

}

else if (dispNr == 1) {

PORTD = 0b00000110;

}

else{

PORTD = 0b00111111;

}

}

void main() {

xNumber = 1 //our x variable

output = 1; //our output of cubic x

TRISD = 0; //output

TRISG = 0; //output

do{

output = xNumber \* xNumber \* xNumber; //

if (output == 9261 ){ //if the 21th number is reached

xNumber =1; //start from the beginning

}

else{

xNumber = xNumber +1; //else update our x value by increasing 1

}

firstDig = modulus(output); //get first digit

output = output / 10;

secondDig = modulus(output); //get second digit

output = output / 10;

thirdDig = modulus(output); //get third digit

output = output / 10;

fourthDig = modulus(output); //get fourth digit

for (slowTemp = 0; slowTemp < 300; slowTemp++){ //run the loop to slow down the result so it becomes visible

setNUM(firstDig); //first nr in 7seg

PORTG = 0x00b0;

Delay\_ms(1);

setNUM(secondDig); //second nr in 7seg

PORTG = 0x0040;

Delay\_ms(1);

setNUM(thirdDig); //third nr in 7seg

PORTG = 0x0002;

Delay\_ms(1);

setNUM(fourthDig); //fourth nr in 7seg

PORTG = 0x0001;

Delay\_ms(1);

}

}while(1);

}