Computer Architecture

Dr. Le Hai Duong & Dr. Ly Tu Nga

Course: IT089IU

Date: March 2021

Time: 6 hours

Name: Nguyễn Anh Khoa

ID: ITITIU19141

Laboratory Session 3

I. Bitwise Logic and Intro. to Procedure (70pts)

1. Exercise 1: (35pts) Write a program that

1.1 Put the number 0xDEADBEEF into register $t1 without using pseudoinstruction li. (lab3_1_1.s)

```
.text

.globl main
main:
ori $t1, $zero, 0×DEAD # $t1 = 0×0000DEAD
sll $t1, $t1, 16 # $t1 = 0×DEAD0000
ori $t1, $t1, 0×BEEF # $t1 = 0×DEADBEEF

jr $ra # Exit
```

1.2 Redo 1.1 as follows: use ori to load each letter into register. (lab3_1_2.s)

```
.text

.globl main
main:
ori $t1, $zero, 0×D # $t1 = 0×0000000D
sll $t1, $t1, 4
ori $t1, $t1, 0×E # $t1 = 0×000000DE
sll $t1, $t1, 4
ori $t1, $t1, 0×A # $t1 = 0×00000DEA
sll $t1, $t1, 4
ori $t1, $t1, 0×D # $t1 = 0×0000DEAD
sll $t1, $t1, 4
ori $t1, $t1, 0×B # $t1 = 0×000DEADB
sll $t1, $t1, 4
ori $t1, $t1, 0×E # $t1 = 0×00DEADBE
sll $t1, $t1, 4
ori $t1, $t1, 0×E # $t1 = 0×0DEADBEE
sll $t1, $t1, 4
ori $t1, $t1, 0×F # $t1 = 0×DEADBEEF

ori $v0, $zero, 10 # EXIT
syscall
```

1.3 Suppose that $t1 = 0xDEADBEEF. Using only register-to-register logic and shift instructions, Reverse the order of the bytes in $t1 so that register $t2 get the bit pattern 0xFEEBDAED. (lab3_1_3.s)

```
.text

.globl main
main:
ori $t1, $zero, 0×DEAD # $t1 = 0×0000DEAD
sll $t1, $t1, 16 # $t1 = 0×DEAD0000
ori $t1, $t1, 0×BEEF # $t1 = 0×DEADBEEF

# I didn't know the we are allowed to use add/addi to make Loop.
# So I try to think of a workaround with nor and sll instructions.
nor $a0, $a0, $zero # set i = 8
jal REVERSE # jump to REVERSE and save position to $ra
```

```
j EXIT # jump to EXIT

EXIT:
ori $v0, $zero, 10 # Exit
syscall

REVERSE:
andi $t3, $t1, 0×F
or $t2, $t2, $t3 # Add the least significant word from $t1 to $t2

sll $a0, $a0, 4 # i -= 1
beq $a0, $zero, JUMP_BACK # if i == 0 then JUMP_BACK

sll $t2, $t2, 4 # Shift $t2 to the left
srl $t1, $t1, 4 # Shift $t1 to the right
j REVERSE # jump to REVERSE

JUMP_BACK:
jr $ra # jump to $ra
```

1.4 Redo 1.3 using only and, or, and rotate instructions. (lab3_1_4.s)

```
.text

.globl main
main:
# Suppose $t1
ori $t1, $zero, 0×DEAD # $t1 = 0×0000DEAD
sll $t1, $t1, 16 # $t1 = 0×DEAD0000
ori $t1, $t1, 0×BEEF # $t1 = 0×DEADBEEF


# Only and, or and rotate instructions.
# Therefore, I didn't know the we are allowed to use add/addi to
make Loop.
# So I try to think of a workaround with nor and sll instructions.

nor $a0, $a0, $zero # set i = 8
jal REVERSE # jump to REVERSE and save position to $ra

j EXIT # jump to EXIT
```

```
REVERSE:
andi $t3, $t1, 0×F
or $t2, $t2, $t3 # Add the least significant word from $t1 to $t2

sll $a0, $a0, 4 # i -= 1
beq $a0, $zero, JUMP_BACK # if i == 0 then JUMP_BACK

ror $t1, $t1, 4 # Shift $t2 to the left
rol $t2, $t2, 4 # Shift $t1 to the right
j REVERSE # jump to REVERSE

JUMP_BACK:
jr $ra # jump to $ra

EXIT:
ori $v0, $zero, 10 # Exit
syscall
```

2. Exercise 2: (15pts) Write a program that

2.1 Set the corresponding bit in register $t1 through $t8. That is, in register $t1 set bit 1, register $t2 set bit 2, and so on. (lab3_2_1.s)

```
.text

.globl main
main:
ori $t0, $t0, 1

sll $t1, $t0, 1
sll $t2, $t1, 1
sll $t3, $t2, 1
sll $t4, $t3, 1
sll $t5, $t4, 1
sll $t6, $t5, 1
sll $t7, $t6, 1
sll $t8, $t7, 1

jr $ra # EXIT
```

2.2 By using ONLY shift instructions and register to register logic instructions (no li pseudoinstruction or addi), put the pattern 0xFFFFFFFF into register $t1. (lab3_2_2.s)

```
.text

.globl main
main:
nor $t1, $zero, $zero

jr $ra # Exit
```

II. MSP430 (30pts)

Step 1: build the sample code in CCS, check the errors.

Step 2: Not run, the values of these registers (PORT_1_2):

P1OUT: **0xBE**

P1IN: **0x06**

P1DIR: **0x00**

P1REN: **0x00**

P1IFG: **0x00**

Step 3: Run, observe and collect the values of these registers in case of

	Red LED ON	**Green LED on**
P1OUT	0xBF = 10111111	0xFE = 11111110
P1IN	0x0F = 00001111	0x4E = 01001110
P1DIR	0x41 = 01000001	0x41 = 01000001
P1REN	0x08 = 00001000	0x08 = 00001000
P1IFG	0xF9 = 11111001	0xF9 = 11111001

Comment and explain the Table above:

From the table above, It is clear that P1OUT, P1IN change while P1DIR, P1REN and P1IFG stay unchanged. P1OUT, P1IN change because they instruct LEDs to switch ON or OFF when the button is pressed.

C Code	MIPS Code

`void main(void) {`
`    WDTCTL = WDTPW | WDTHOLD;`
	c000: 40B2 5A80 0120 MOV.W #0x5a80,&Watchdog_Timer_WDTCTL

```
// stop watchdog timer

    P1OUT |= Red;
    P1OUT &= ~Green;
    P1DIR |= Red +Green;

    P1DIR &= ~Button;
    P1REN |= Button;
    P1OUT |= Button;

    volatile unsigned int i;
// volatile to prevent optimization

    while(1)
    {
        if ((P1IN & Button)!= Button)
        {
            while ((P1IN & Button)!= 
Button)
            {

            }
            P1OUT ^= Red + Green;
        }
    }
}
```

```
10       P1OUT |= Red;
c006:   D3D2 0021          BIS.B
#1,&Port_1_2_P1OUT
11       P1OUT &= ~Green;
c00a:   F0F2 00BF 0021      AND.B
#0x00bf,&Port_1_2_P1OUT
12       P1DIR |= Red +Green;
c010:   D0F2 0041 0022      BIS.B
#0x0041,&Port_1_2_P1DIR
14       P1DIR &= ~Button;
c016:   C2F2 0022          BIC.B
#8,&Port_1_2_P1DIR
15       P1REN |= Button;
c01a:   D2F2 0027          BIS.B
#8,&Port_1_2_P1REN
16       P1OUT |= Button;
c01e:   D2F2 0021          BIS.B
#8,&Port_1_2_P1OUT
22            if ((P1IN & Button)!= Button)
     $C$L1:
c022:   B2F2 0020          BIT.B
#8,&Port_1_2_P1IN
c026:   23FD               JNE     ($C$L1)
24               while ((P1IN & Button)!=
Button)
     $C$L2:
c028:   B2F2 0020          BIT.B
#8,&Port_1_2_P1IN
c02c:   27FD               JEQ     ($C$L2)
28               P1OUT ^= Red + Green;
c02e:   E0F2 0041 0021      XOR.B
#0x0041,&Port_1_2_P1OUT
c034:   3FF6               JMP     ($C$L1)
 85   {
     _c_int00_noinit_noargs_noexit():
c036:   4031 0400          MOV.W
#0x0400,SP
 87      _system_pre_init();
c03a:   12B0 C056          CALL
#_system_pre_init
 88      main(0);
c03e:   430C               CLR.W   R12
c040:   12B0 C000          CALL    #main
 89      abort();
c044:   12B0 C050          CALL    #abort
48            BIS.W     #(0x0010),SR
     $isr_trap.asm:48:59$(),
__TI_ISR_TRAP():
c048:   D032 0010          BIS.W
#0x0010,SR
```

	49 JMP __TI_ISR_TRAP c04c: 3FFD JMP ($isr_trap.asm:48:59$) 51 NOP ; CPU40 Compatibility NOP c04e: 4303 NOP 100 { C$$EXIT(), abort(): c050: 4303 NOP 108 for (;;); /* SPINS FOREVER */ $C$L1: c052: 3FFF JMP ($C$L1) c054: 4303 NOP 58 return 1; _system_pre_init(): c056: 431C MOV.W #1,R12 c058: 4130 RET

Explain:

From line 8-16, we set up the variables for LEDs and Button. **"P1OUT |= Red**;" Red LED is On and **"P1OUT &= ~Green;"** Green LED is Off.

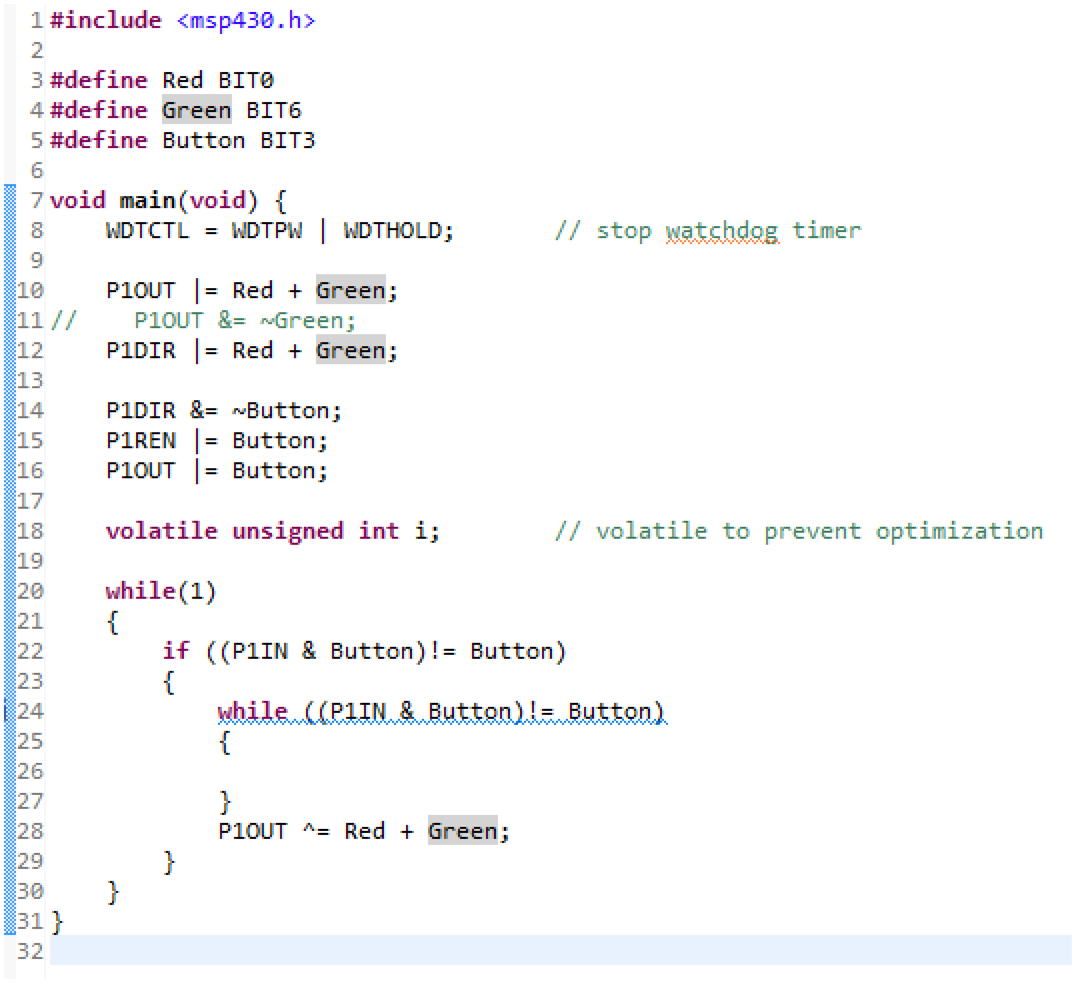
Then we enter the while loop **"While (1)"** for checking and updating every button pressed.

"If (P1IN & Button)!= Button" detects when the button is pressed.

"while ((P1IN & Button)!= Button)" waits for the button to be release then switches the LEDs between Red and Green **"P1OUT ^= Red + Green;"**.

Problem 1: modify the sample code in order to when pressing the button two LEDs turn on and vice versa.

Nguyễn Anh Khoa – ITITIU19141



```c
#include <msp430.h>

#define Red BIT0
#define Green BIT6
#define Button BIT3

void main(void) {
    WDTCTL = WDTPW | WDTHOLD;       // stop watchdog timer

    P1OUT |= Red + Green;
//    P1OUT &= ~Green;
    P1DIR |= Red + Green;

    P1DIR &= ~Button;
    P1REN |= Button;
    P1OUT |= Button;

    volatile unsigned int i;        // volatile to prevent optimization

    while(1)
    {
        if ((P1IN & Button)!= Button)
        {
            while ((P1IN & Button)!= Button)
            {

            }
            P1OUT ^= Red + Green;
        }
    }
}

```