Computer Architecture Course: IT089IU

International University – VNU HCM Date: March 2021

Dr. Le Hai Duong & Dr. Ly Tu Nga Time: 6 hours

**Laboratory Session 2**

# **I. Testing and Branching (70pts)**

1. **MIPS assembler directives**

A screenshot of a cell phone

Description automatically generated

Figure 1 MIPS Memory Usage as viewed in SPIM

**.text**

indicates that following items are stored in the user text segment, typically instructions

**.data**

indicates that following data items are stored in the data segment

**.globl** sym

declare that symbol sym is global and can be referenced from other files

**Common data definitions**

**.word** w1, ..., wn

store n 32-bit quantities in successive memory words

**.half** h1, ..., hn

store n 16-bit quantities in successive memory halfword

**.byte** b1, ..., bn

store n 8-bit quantities in successive memory bytes

**.ascii** str

store the string in memory but do not null-terminate it

* strings are represented in double-quotes “str”
* special characters, eg. \n, \t, follow C convention

**.asciiz** str

store the string in memory and null-terminate it

**.float** f1, ..., fn

store n floating point single precision numbers in successive memory locations

**.double** d1, ..., dn

store n floating point double precision numbers in successive memory locations

.**space** n

reserves n successive bytes of space

.**align** n

align the next datum on a 2n byte boundary.

For example, **.align 2** aligns next value on a word boundary.

.**align 0** turns off automatic alignment of **.half**, **.word**, etc. till next **.data** directive

1. **Pseudo instructions (20pts)**

Pseudo intructions do not correspond to real MIPS instructions. Assembler would translate pseudo intructions to real instructions (one or more instructions). Pseudo instructions not only make it easier to program, it can also add clarity to the program, by making the intention of the programmer more clear.

* 1. Load the assembly file **Lab2\_2.s** into qtSpim and run. What is the **real** instruction shown in qtSpim for the pseudo instruction “**li $t0, 5**”?
  2. Change the pseudoninstruction “**li $t0, 5**” in **Lab2\_2.s** to “**li $t0, -5**”. What are the real MIPS instructions for “**li $t0, -5**”. Explain how the real instructions work.
  3. Change the pseudoninstruction “**li $t0, 5**” in **Lab2\_2.s** to “**li $t0, 0xaabbccdd**”. What are the real instructions for “**li $t0, 0xaabbccdd**”. Explain how the real instructions work.

1. **Branching (35pts)**
   1. Load the assembly file **Lab2\_3.s** into qtSpim and run. Try to win the game. What is the **secret number**?
   2. The figure 2 shows the output when player wins the game.

A screenshot of a cell phone

Description automatically generated

The assembly print both win and lose results. Fix the assembly and save it as **Lab2\_3.2.s**

* 1. Modify the game so that it will print out as follow (no iteration) using the instructions **bgt,** or **bge,** or **blt,** or **ble**:

A screenshot of a cell phone

Description automatically generated

Save your file as **Lab2\_3.3.s**

* 1. Modify the game so that player can keep guessing until he find the secret number. Save your assembly as **Lab2\_3.4.s**
  2. Modify previous version so that player can decide to stop the game by input a **flag**. Save your assembly as **Lab2\_3.5.s**

1. **String (15pts)**

Modify **Lab2\_4.s** so that it converts an input string as follow:

A screenshot of a cell phone

Description automatically generated

Assume that the input string contains all lowercase letters. The first letter of every word is capitalized. Save your assembly as **Lab2\_4\_1.s**

# **II. MSP430 (30pts)**

Given a sample code to control LED via pushing button in MSP430 as follows

|  |  |  |
| --- | --- | --- |
| **No.** | **Sample codes** | **Comments/Results/Functions** |
| **1.**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22**  **23**  **24**  **25**  **26**  **27**  **28**  **29** | **#include** <msp430.h>  **#define** Red BIT0  **#define** Green BIT6  **#define** Button BIT3  **void** **main**(**void**) {  WDTCTL = WDTPW | WDTHOLD;  P1OUT |= Red;  P1OUT &= ~Green;  P1DIR |= Red +Green;  P1DIR &= ~Button;  P1REN |= Button;  P1OUT |= Button;  **while**(1)  {  **if** ((P1IN & Button)!= Button)  {  **while** ((P1IN & Button)!= Button)  {  }  P1OUT ^= Red + Green;  }  }  } |  |

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

**Step 2:** **Not run**, the values of these registers (PORT\_1\_2):

P1OUT:

P1IN:

P1DIR:

P1REN:

P1IFG:

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

|  |  |  |
| --- | --- | --- |
|  | **Red LED On** | **Green LED On** |
| P1OUT |  |  |
| P1IN |  |  |
| P1DIR |  |  |
| P1REN |  |  |
| P1IFG |  |  |

Comment and explain the Table above: **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

When running and pausing, click View and open ***Disassenbly*** window, write down **these instructions of sample code:**

|  |  |  |
| --- | --- | --- |
| **No.** | **C code** | **MIPS code** |
| **1.**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22**  **23**  **24**  **25**  **26**  **27**  **28**  **29** |  |  |

Based on the Table above, **please explain the process of sample code under if-then else conditional statements:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

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

**Your code:**

**Reference:**

1. <https://en.wikibooks.org/wiki/MIPS_Assembly/Pseudoinstructions>
2. <https://courses.missouristate.edu/KenVollmar/MARS/Help/SyscallHelp.html>
3. <https://www.assemblylanguagetuts.com/mips-assembly-programming-tutorials/#MIPS_Data_Types>
4. <https://en.wikibooks.org/wiki/MIPS_Assembly/Arithmetic_Instructions>
5. <https://gab.wallawalla.edu/~curt.nelson/cptr280/lecture/mips%20arithmetic%20instructions.pdf>