TRC2400 Computer Programming
ECE2071 Computer Organisation and Programming

Laboratory Session 9
Week 10– Semester 1 2011

IMPORTANT – MARKING
You will receive marks for preliminary work and lab completion by completing quizzes on Blackboard
All quizzes receive equal marks and these will be scaled to give a final lab mark worth 10% of your final assessment.
You MUST complete the preliminary work quiz BEFORE midnight of the day before your lab otherwise you will receive a zero mark for the lab exercise (both preliminary and completion mark)
You must start the completion quiz before the end of your laboratory period (you will need the demonstrator to enter a password which will only be provided when you complete the lab)

Introduction to the stack and sentinel controlled loops in MIPS assembly language programs

1. Objectives
To gain familiarity with the following:

1. Interacting with external events by testing the status of input ports,
2. Managing the stack in MIPS assembler programs

2. Preliminary work
Before coming to the lab you should complete the preliminary work quiz on Blackboard. This week's quiz will cover your understanding of the exercises in this laboratory exercises. In particular, you should be familiar with the simple machine language instructions in the course literature. You should refer to the attached MIPS reference data summary sheet to help understand the various MIPS assembler instructions used within the lab exercise programs. This MIPS reference data summary sheet should be brought along to the lab to help you write and understand existing MIPS assembly language programs.

3. First exercise
In one of last week's exercises you modified a program that displayed bit patterns on the I/O port LEDs. In this first exercise you should create a new project called Lab9_1 and within the project an assembler file called Lab9_1.s. Copy the assembler code for exercise Lab8_2 (without the errors) into your assembler file. The objective of this exercise is to modify the Lab8_2 code so that one LED lights and each time the program goes through the loop the next LED is illuminated and the previous extinguished. When the 5th LED is illuminated then the 0th should be the next one in sequence. So the LEDs should light in the following sequence:

7th	6th	5th	4th	3rd	2nd	1st	0th
OFF	OFF	OFF	OFF	OFF	OFF	OFF	ON
OFF	OFF	OFF	OFF	OFF	OFF	ON	OFF
OFF	OFF	OFF	OFF	OFF	ON	OFF	OFF
OFF	OFF	OFF	OFF	ON	OFF	OFF	OFF
OFF	OFF	OFF	ON	OFF	OFF	OFF	OFF
OFF	OFF	ON	OFF	OFF	OFF	OFF	OFF
OFF	OFF	OFF	OFF	OFF	OFF	OFF	ON
OFF	OFF	OFF	OFF	OFF	OFF	ON	OFF
And so on …..							

Hint: multiplying a binary number by 2 is the same as shifting the bits one place to the left and adding a binary number to itself is the same as multiplying it by 2.

4. Second exercise

Based on your solution to the first exercise in this exercise you are going to complete the programming of an electronic dice. Create a new project Lab9_2 together with an assembler file Lab9_2.s. Copy your code for exercise 1 and insert code into the loop which checks the state of switch number 3. When the switch is OFF the program loop should cycle as in the first exercise. When the switch in ON the program should stay in a loop testing the switch and only proceeding when it is OFF again (this will freeze the state of the LEDs). Thus when the switch is OFF the LEDs will cycle very rapidly – rolling the dice. When the switch is ON the LED display will freeze showing the illuminated LED in one of 6 positions (indicating the number on the dice).

5. Third exercise

Create a new project called Lab9_3 and associated assembler file Lab9_3.s for this exercise. On Blackboard you are provided with the following skeleton code that is part of a program that calculates the square of a number using a recursive function. Copy the contents of the Blackboard file containing this code into your Lab9_3.s file.

```
# Laboratory Exercise 3 for lab 9
# Written by Andy Russell 20/03/2009

       .set noreorder
       .text
       .globl start
       .ent start

start:  lui $29, 0x8001              # Load $sp with upper half of stack address
                                     # Lower half is filled with zeros
       ori $29, 0xFFF8               # Now load the lower 16 bits
       li  $4, 0x0003                # load argument register $a0 with number to be squar
       jal square                    # call the recursive function to calculate the
square
       nop
infinite: b infinite                 # wait here when the calculation has finished
                                     # $v0 = $2 contains the result
       nop                           # Needed after branch

#-----------------------------------------------------------------
#square - input parameter $a0 contains number to be squared
#        result returned in register $v0
#-----------------------------------------------------------------
square: sub $29, $29, 8              # decrement the stack pointer $sp
       sw  $31, 4($29)               # push the return address register $ra
       nop                           # another nop
       sw  $4, 0($29)                # push argument register $a0
       nop                           # yet another nop
       li  $8, 0x0001                # load $t0 with 1 as part of test for base case
       bne $4, $8,notbase            # branch if not the base case
       nop
       li  $2, 0x0001                # return base result in $v0
       add $29, $29, 8               # recover stack space (Note: did not corrupt
registers)
       jr     $31                    # jump to return address in $ra
       nop
notbase:       #****************************************
               #your code for the non-base case goes here
               #****************************************
       jr  $31                       # jump to contents of return address register $ra
       nop
.end start                           # Marks the end of the program
```

The recursive function is based on the following equation:

$$n^2 = (n - 1)^2 + 2 * (n - 1) + 1$$

This equation is ideal for implementation on simple processors because it only uses addition if $(n - 1)^2$ is available.

The base case is: $1^2 = 1$

If the input value n is not the base case then the recursive function must calculate $2 * (n -1) + 1$ and then add $(n - 1)^2$ which is found by a recursive call to itself. Note that on entry your recursive function must push $ra and $a0 onto the stack and pop the variables when exiting the function. Your task for this exercise is to provide code for the recursive function that implements the non-base case.

6. Conclusion

Please note that marks will not be allocated to people who do not attend their allocated lab and complete the appropriate quizzes by their deadline. Under no circumstances will marks be recorded after the laboratory period is finished.

RAR 20/3/2009; RAR 1/3/2011