TRC2400 Computer Programming
ECE2071 Computer Organisation and Programming

Laboratory Session 8

Week 9– 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 MIPS Assembly language programs and MipsIt

1. Objectives
To gain familiarity with the following :

1. MipsIt program development & simulation tool,
2. MIPs reference data summary sheet, and
3. interpreting existing MIPS programs.

After this laboratory exercise, you should understand how a computer executes simple machine language instructions, and how instructions and data are stored in memory. You should also be acquainted with the MipsIt environment, the lab computer, the MIPS simulator and the MIPS reference data summary sheet.

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 exercise. 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.

For your own use you may download the MipsIt code used in this lab from a number of locations on the web. A web search with the search term "mipsit software" will produce a number of suitable university sites.

3. First exercise
In this exercise you will enter the following assembler language program into MipsIt and run it using the MIPS simulator. Note that this program is available on Blackboard, do not cut and paste from the pdf file.

```
# Laboratory Exercise 8_1
# Based on code written by Jan Eric Larsson, 27 October 1998

.set noreorder                    # Avoid reordering instructions
.text                             # Start generating instructions
.globl start                      # This label should be globally known
.ent start                        # This label marks an entry point

start:  li $8, 0x1                # Load the value 1
```

```
                li $9, 0x2              # Load the value 2
                add $10, $8, $9         # Add the values
infinite:
                b infinite              # an infinite loop so that the
                                        # computer will not execute code
                                        # past the end of the program
                nop                     # all branch and lw, sw must
                                        # be followed by a nop
.end start                              # Marks the end of the program
```

Step 1: Start the MipsIt program by double clicking on the MipsIt icon on the desktop:

MipsIt
MipsIt

Ignore the warning about being unable to connect to a serial port. This is associated with add on hardware that you will not be using. Create a new assembler project called Lab8_1 and then create a new assembler file ***File-New-File (assembler)*** "lab8_1.s", cut and past the program supplied on Blackboard and save the file. Note that you are asked where to save the file when you create it. There must be no spaces in the path names – for example "Program Files" is not suitable because of the space between "Program" and "Files".

Step 2: Translate the program to machine code, using the Build command in the Build menu, or by pressing the button F7. If the assembler finds any syntax errors, it will tell you on which line the first error is located. Correct all errors and rebuild the program.

Step 3: Start the MIPS simulator by double clicking on the Mips.exe icon:

Mips
Mips
Lunds Universitet

Study the simulator and identify the main parts. It is possible to inspect the contents of the registers, program counter and memory by selecting the appropriate items under the **View** tab.

Step 4: Upload your compiled program to the simulator from MipsIt by selecting **Build-Upload-To Simulator** (or by pressing F5) and you should find that the code has appeared in memory of the MIPS simulator at location 0x80020000.

Step 5: A disassembler is a program that converts machine code instructions stored in memory back to assembler instructions. Inside the MIPS simulator Memory window inspect the loaded instructions. Compare the disassembled code with the original code in your assembly file.

Step 6: The simulator will execute a program, instruction by instruction, when you use the command ***step (icon -- blue box with arrow above)***. Use this command and execute the program step by step, and inspect what happens with the registers (*click on CPU block to display*) and program counter after each step. It is a good idea to work out what each instruction will do before executing it and then check that the simulator does what you predicted.

Note: that sometimes the upload is not performed correctly. If the program does not seem to be working correctly in the simulator you could try reloading it to see if that helps.

Step 7: It is also possible to let the computer execute a program at full speed and stop when the program counter contains a certain address, set a breakpoint at the add instruction by doing the following. Reset the simulator by re-uploading the program from MipsIt and set a breakpoint at the add instruction. Do this by *right clicking on the line where the breakpoint is required and selecting set breakpoint*. Then start the program using the command ***green arrow***. Investigate what happens to the registers and program counter. What would have happened if there was no breakpoint?

4. Second exercise

The following assembly program reads the position of the eight switches and uses this to control the eight corresponding LEDs on the simulator computer I/O board (**View-I/O** in the MIPs simulator). First, the program loads the address of the input port, 0xbf900000, into register $9. The address of the output port is the same as the input port. Then the program reads eight bits (one byte) from the input port, where each bit corresponds to the position of one switch. It then writes the eight bits to the output port, which controls the LED display. Finally, it branches back to the repeat label and starts the read/write cycle over again.

```
# Laboratory Exercise 8_2
# Written by Jan Eric Larsson, 27 October 1998

       .set noreorder
       .text
       .globl start
       .ent start

start: lui 9, 0xbf90             # Load upper half of port address
                                 # Lower half is filled with zeros
repeat lbu $8, 0x0($9)           # Read from the input port
       nop                       # Needed after load
       sb $8, 0x0($9)            # Write to the output port
       b repeat                  # Repeat the read and write cycle
       nop                       # Needed after branch
       li $8, 0                  # Clear the register
.end start                       # Marks the end of the program
```

Create a new project (lab8_2), cut and paste the code from the file available on Blackboard. Save it in a new file (lab8_2.s). Unfortunately, this assembler code contains errors. Look at the compiler errors that are produced when you try to **Build-Rebuild All** and correct them. When the program is compiling correctly load it into the simulator. Execute the program step by step and investigate what happens to the registers and program counter. Then flip a switch (*click on I/O block*) and investigate what happens. When do the LEDs get turned on and off? When you understand how the program works, you can run it at full speed and flip one or several switches.

5. Third exercise

Add code to the program you got working in exercise two so that the first two LEDs are always on and the next two LEDs are always off independent of the state of the switches.

6. Fourth exercise

Write an assembler program (Lab8_4.s) to multiply two numbers together (in $8 and $9) and to place the result in $7. The multiplication should be performed by successive addition. The following C code should give you the idea:

```
a=0;            // result accumulated in a, initially it is zero
b=2;            // multiplier
c=3;            // multiplicand
do{
    a=a+b;
    c=c-1;
} while (c>0);
```

compile your code and make sure that is works as expected.

7. 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 14/3/2009; RAR 1/3/2011