Laboratory Exercise 1
Machine Language Instructions

Goals

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, and the MIPS simulator.

Literature

- Patterson and Hennessy: Chapters 2.1–2.4, 2.9–2.10, Appendix A.10, or
  Brorsson: Chapter 2, Appendix D.
- MIPS Lab Environment Reference.

Preparation

Before you start the exercise, you should read about simple machine language instructions in the course literature and solve the home assignments below. You should also read the entire laboratory exercise in detail.

You will find the MipsIt programming environment referred to in this laboratory exercise on the companion CD; see the Software page on the CD if you have not already installed this software. It will be convenient for you to create shortcuts for the files MipsIt.exe and Mips.exe on the desktop. Create a new project file for each new program. Save the old project before creating a new one.

Home Assignment 1

Study the following small assembly program and explain how it works. The first four lines are directives that tell how the program should be translated by the assembler. You can ignore them for now. The three lines starting with the label `start` produce actual machine instructions. Use the literature to explain what these instructions mean and what the program does when executed. Note that the instruction `li $8, 0x1` is a pseudo-instruction that is replaced by the actual instruction `addiu $8, $0, 0x1` before it is translated to machine code. Also note the comments after #. From now on, all the programs that you write for this course should be properly commented.

```
        # Laboratory Exercise 1, Home Assignment 1
        # Written by Jan Eric Larsson, 27 October 1998

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

start:  li      $8, 0x1     # Load the value 1
        li      $9, 0x1     # Load the value 1
        add     $10, $8, $9 # Add the values

        .end start          # Marks the end of the program
```

Home Assignment 2

The following assembly program reads the position of the eight switches and controls the eight corresponding LEDs on the lab computer board. First, the program loads the address of the input port, `0xbf900000`, in 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 1, Home Assignment 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
```

Translate this program to hexadecimal machine instructions, using the literature. Note that the pseudo-instruction `b repeat` is performed by the actual instruction `beq $0, $0, repeat`. The instructions `lbu` and `b` must be followed by the instruction `nop`, which is represented as all zeros.

A Small Machine Language Program

The machine language instructions are the simplest actions that the computer can perform. They can be combined to perform more complex operations. A part of the assembly program of Home Assignment 1 is shown below.

```
start:  li      $8, 0x1     # Load the value 1
        li      $9, 0x1     # Load the value 1
        add     $10, $8, $9 # Add the values
```

This program performs the simple addition 2 = 1 + 1. First, it loads the value 1 in registers $8 and $9, and then it adds the contents of these registers and puts the result in register $10. The label `start` marks a symbolic address. It can be used to refer to an address without actually knowing its real value.

Assignment 1

Start the MipsIt programming environment. Create a new assembler project. Create a new assembler file, type in the program of Home Assignment 1 and save the file.

Assignment 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 syntactic errors, it will tell you on which line the first error is located. Correct all errors and rebuild the program.

Assignment 3

Study the lab computer and equipment, and identify the main parts. Make sure that the lab computer is connected to the power supply and to the host computer. Press the reset button on the lab computer.

It is possible to inspect the contents of the registers and the program counter. The command *dr* (written in the console window) prints the names and contents of all registers, and the command *dr pc* prints the contents of the program counter. Initialize the registers using the *init* command and look at the contents of the registers and the program counter using the *dr* and *dr pc* commands. The console window is opened by the `Console` command in the `View` menu, or with the button `F9`.

Assignment 4

Now upload the machine language program to the lab computer. Use the command Upload to Hardware in the Build menu. The instructions will be uploaded beginning at the address `0x80020000` in the lab computer memory.

Assignment 5

A disassembler is a program that interprets memory contents as instructions. Use the disassembler in the lab computer to inspect the uploaded instructions. You can do this by typing the command d*is 0x80020000* in the hardware window. Compare the disassembled code with the original code in your assembly file.

Assignment 6

The lab computer will execute a program, instruction by instruction, when you give the command *step* in the console window. Use this command and execute the program step by step, and inspect what happens with the registers and program counter after each step.

Assignment 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, a *breakpoint*. Reset the contents of the registers to zero, with commands such as *fr r8 0x0*, and set a breakpoint immediately after the last address of the program, using the command *b address*. Then start the program using the command *go start*. Investigate what happens to the registers and program counter. What would have happened if there was no breakpoint?

Assignment 8

Add a branch instruction that will cause an endless loop. Add the instruction `b start` at the end of the program (followed by a `nop` as usual), assemble, and upload it again. What happens when you start the new program? How can you stop it?

Communication with the Environment

The first program above only uses internal registers. However, the program of Home Assignment 2 reads and writes on ports connected to the real world.

Assignment 9

Create a new project, type in the program of Home Assignment 2 and save it in a new file. Build it and upload it to the lab computer.

Execute the program step by step and investigate what happens to the registers and program counter. Then flip a switch and investigate what happens. When you understand how the program works, you can run it at full speed and flip one or several switches. Explain all the things that happen when a switch is flipped.

Instruction Formats and Addressing Modes

In this part of the laboratory exercise, you will use a software simulator for the MIPS computer. Start the MIPS simulator. Identify the different parts that correspond to the physical parts of the lab computer board.

You will now study a machine instruction in detail, and see how it can be translated from assembly to machine code, how it is stored in memory, and how it can be executed. Start with the single instruction:

```
addi    $8, $8, 0x1
```

This instruction adds the contents of register $8 with the value 1 and places the result in register $8. (The old contents are overwritten.) The instruction is shown in assembly format. It cannot be executed as is by the computer, but must be translated to machine code.

Assignment 10

Use the literature to figure out how many bits that are needed to represent the instruction in machine language. Then translate the instruction into machine code. Give your answer as a hexadecimal number.

Assignment 11

Type in the machine instruction above into the simulator memory at the address `0x80020000`, and set the program counter to `0x80020000`. Then check that your translation was correct by using the simulator disassembler.

The simulator disassembler can show memory contents as integers, floating point numbers, ASCII text, or as instructions. What is it that really is stored in the memory? Write down the memory contents at the address `0x80020000` as an integer, as a floating point number, and ASCII, and as an instruction. How many bits did you look at?

How can the computer know that the stored bit pattern is an instruction? What is the ASCII equivalent of the instruction?

Assignment 12

Set the value of register $8 to `0xff`. Execute the instruction using the command Step in the Cpu menu, or by clicking the `step` button in the tool bar. Investigate what happens to the registers and program counter. What is the new value of the program counter? Why? What is the new value of register $8? Why?

Addressing Modes

The instruction above contained the value of the rightmost operand in the instruction code itself. This is called *immediate addressing*. Now consider the instruction:

```
add     $8, $8, $9
```

Assignment 13

Translate the instruction above to machine code and enter it in memory at the next available address. Verify that your translation was correct with the disassembler. Set register $8 to 1 and $9 to 2 and execute the instruction. Investigate the result. What does register $8 contain after the execution? Why? This addressing mode is called *register addressing*.

Assignment 14

Finally, consider the following instruction:

```
lw      $8, 0x0($9)
```

Translate this instruction to machine code, enter it at the next available address. Set register $9 to contain the address of a word where you already know the memory contents, then execute the instruction. Explain the new value of register $8. Explain in detail all the parts of the address of the instruction. This addressing mode is called *base* or *displacement addressing*.

Simulator versus Hardware

As you have seen, the lab equipment includes a physical lab computer as well as a simulator. The simulator software can be downloaded from the course web site to your home computer. Thus, if you have a PC compatible computer, you can prepare all laboratory exercises at home in advance.

Assignment 15

Enter the hexadecimal code of Home Assignment 2 into the simulator, and verify that it works as expected.

Assignment 16

Upload the two programs from the home assignments to the simulator and verify that the lab computer and the simulator both work in the same way.

Conclusions

Before you finish the laboratory exercise, think about the questions below:

- Where are the programs you have used executed?
- How can the computer jump to a symbolic label, such as occurs in the instruction`b repeat`?
- Which registers are affected by a branch instruction?
- How many bytes are used to store one instruction?
- Can the computer execute an assembly instruction?
- How does the computer know that a bit pattern is an instruction?