Introduction

Objectives

At the end of this lab you should be able to:

- Explain how common program variables are stored
- Distinguish between different types of high-level program statements
- Understand low-level code corresponding to program statements
- Explain how program subroutines work
- Use the simulator to create user interrupts

Processor (CPU) Simulators

The computer architecture tutorials are supported by simulators, which are created to underpin theoretical concepts normally covered during the lectures. The simulators provide visual and animated representation of mechanisms involved and enable the students to observe the hidden inner workings of systems, which would be difficult or impossible to do otherwise. The added advantage of using simulators is that they allow the students to experiment and explore different technological aspects of systems without having to install and configure the real systems.

Basic Theory

High-level language (HLL) programs are made of variables holding data values and multiple program statements as algorithms. These statements often control the flow of program execution under certain conditions. Calls to subroutines and interrupts all change the sequential flow of a program execution without which feature programs would not do any useful work.

Simulator Details

This section includes some basic information on the simulator, which should enable the students to use the simulator. The tutor(s) will be available to help anyone experiencing difficulty in using the simulator.

The simulator for this lab is an application running on a PC and is composed of multiple windows.

🔍 CPU Simulator: CPU0										
INSTRUCTION MEMORY (BAM)	EXECUTION UNIT	SPECIAL REGISTERS REGISTER SET								
PAdd LAdd Instruction Base	1 FETCH	PC 22	Reg Value C Val 🔺							
0000 0000 MOV #0, R08 0000	JGI 48	IB TOT 49	□R00 0							
00006 0006 MOV #1, R09 0000	2 DECODE On Code JGT	JG1 48	□R01 0							
00012 0012 CMP #20, R09 0000		SR 2	□R02 0							
00018 0018 JGT 48 0000		SP 0	□R03 1							
00022 0022 MOV R08, R03 0000			R04 0							
00027 0027 ADD #1, R03 0000		BR <mark>0</mark>	□R05 0							
00033 0033 MOV R03. R08 0000		MAR 18	□R06 0							
00038 0038 ADD #1. R09 0000		Chatua Elaga								
□0044 0044 JMP 12 0000	3. EXECUTE SHOW CACHE									
00048 0048 HLT 0000			LIR09 2							
	PROGRAM LIST	HARDWARE STACK								
	Name Base Start									
	FORNEXT 0000 0000									
	SHOW PROG MEMORY		$\square B19 0$							
	RESET SHARE DELETE		□R21 0							
			□R22 0 +							
	REMUVE PRUG CLEAR ALL		Registers							
PROGRAM INSTRUCTIONS	PROGRAM CONTROL	ADVANCED	2 CHANGE RESET ALL							
ADD INSERT INSERT DELETE EDIT	Court	COMPILER OS								
NEW ABOVE BELOW	STEP Speed		Watch Parameters:							
Program Name File	Fast	INPUT/OUTPUT	= T SET WATCH RESET							
SAVE FORNEXT V	RUN		- FYIT -							
Base Address () Base Address			Copyright @ 2006							
ADD LOAD	STOP - J Slow	NEW CPU Select	Besim Mustafa CLOSE							
			Luge mit university							

Image 1 - Main simulator window

The main window shown in Image 1 is composed of several sub-views, which represent different functional parts of the simulated processor. For this lab session we are interested only in the compiler part of the simulator.

ADVANCED		2			
COMPILER	OS 0	VIRTUAL OS			
INPUT/OUTPUT		INTERRUPTS			
NEW CPU	Select	CPU 0 💌			
ENDIANNES:	⊙ LIT	TLE C BIG			

Image 2 - Advanced functions

In order to access the compiler, click on the **COMPILER...** button as shown in Image 2 on the right. The compiler window shown in Image 3 below will show.

🐃 Program Compiler							
PROGRAM SOURCE (INPUT)	POGRAM CODE (OUTPUT)						
NoName	NoName						
NEW CLEAR CLOSE GO TO FIND -> Col 1 Line 1 COMPILER PROGRESS	Add	Instruction	Code	Lin	Comments		
COMPILER Fast Compilation Fast Compilation SHOW SUBROUTINE LIST. Defining Required All Constart Folding Fast Constart Folding Fast Constart Folding Fast Fast Fast Fast Fast Fast Fast Fast	CODE Start Address Size GOTO OS.	Base Address	□ Include source CODE SHOW EXIT Copyright @ Besim M Edge Hill C	PBI PBI 2006-2008 ustafa Jniversity	NT LOAD JDE SAVE CLOSE		

Image 3 - The main compiler window

In the compiler window there are three main sub-windows

- **Program Source** all high-level source statements appear here.
- **Compiler Progress** information on the progress of a compilation appear here.
- **Program Code** assembly code generated by the compiler appear here.

Lab Exercises - Investigate and Explore

The lab exercises are a series of experiments, which are attempted by the students under guidelines. The students are expected to carry out further investigations on their own in order to form a better understanding of the technology.

Now, have a go at the following activities:

1. Enter the following source code and compile it.

```
program Test1
   var IntVar integer
   var BoolVar boolean
   var StrVar1 string (5)
   var StrVar2 string(20)
   IntVar = 6
   BoolVar = true
   StrVar1 = "Hello"
   StrVar2 = "And again"
end
```

Now click on the **SYMBOL TABLE...** button. The **Symbol Table** window will show. Observe the kind of information kept in the symbol table. Make a note of the **type**, **size** and **address** fields for each of the entries in the table.

Next, load the compiled program in memory. In the CPU simulator window click on the **SHOW PROG MEMORY...** button. The contents of the program data memory will show. Make sure it stays on top. Then run the program at maximum speed. Observe the contents of the memory paying attention to the address locations you noted before.

2. Enter the following source statements

```
Program Test2
    n = 0
    i = n + 1
    p = i * 3
    writeln (" n=", n, " i=", i, " p=", p)
end
```

Compile the above program. Now observe the code generated in the **PROGRAM CODE** window. You don't need to analyse it in detail. However, count the number of jump instructions (i.e. those that start with a letter 'J') and note this down. Can you tell what kind of program statements this program is using?

3. Enter the following source statements

```
Program Test3
    n = 0
    if n < 5 then
        p = p + 1
        end if
end</pre>
```

Compile the above program. Now observe the code generated. How many jump instructions are there? What do you think is the purpose of the jump instruction in this code? What kind of a statement is an '**if**' statement?

4. Enter the following source statements

```
program Test4
    p = 1
    for n = 1 to 10
        p = p * 2
        next
end
```

Compile the above program. Now observe the code generated. How many jump instructions are there? What do you think is the purpose of each of the jump instructions in this code? What kind of a statement is a '**for**' statement? Can you think of another statement of this kind (you can give an example from any programming language you are familiar with)?

5. Enter the following source statements

```
Program Test5

sub One

writeln("I am sub One")

end sub

sub Two

call One

writeln("I am sub Two")

end sub

call Two

End
```

Compile the above program. Next, load the compiled program in memory. In the CPU simulator window click on the **SHOW PROG MEMORY...** button. Click on the **SHOW PIPELINE...** button and check the checkbox labelled **No instruction pipeline**. Close the window. In the CPU simulator window do the following

Click on the **RESET** button in **PROGRAM LIST** frame. Now you'll manually execute this program instruction by instruction. To do this double-click the currently highlighted instruction. So, you'll start with the **MSF** instruction, and then do the **CAL** instruction, etc. As you execute the program in this manner, make the following observations: make a

note of the **PROGRAM STACK** frame contents after executing a **CAL** instruction or a **RET** instruction. Keep executing instructions until you reach the **HLT** instruction.

Can you explain what is happening each time a **CAL** or a **RET** instruction is executed and how they affect the **PROGRAM STACK** contents.

6. Enter the following source statements

```
program Test6
   sub Any
        n = 0
   end sub
   sub MeToo intr 5
        writeln("Me too, me too!")
   end sub
   do
   loop
end
```

Compile the above program. Look at the code generated. What address does subroutine "MeToo" start at? Make a note of this number. Next, load the compiled program in memory. In the CPU simulator window click on the **INTERRUPTS...** button. The **Interrupts** window will show. Make a note of the interrupt number against which a number appears in the corresponding box. Do these numbers mean anything to you? Explain.

Make sure the **Interrupts** window stays on top. Click on the **INPUT/OUTPUT...** button and make sure the **Console** window also stays on top. Now slide the speed of the CPU simulator to the fastest speed and run the program. Make a note of what you are observing. What is the main purpose of the "**do**" loop statement in this program?

Next, click on the **TRIGGER** button in the **Interrupts** window while at the same time you keep your eye on the **Console** window. Make a note of what you are observing.

Slow down the CPU simulation (e.g. a little above half way on the sliding scale). Trigger the interrupt and observe the **PROGRAM STACK** contents. You can click on the **STOP** button as soon as you see numbers appearing on this stack so that you have time to look at its contents. What do you observe?

There are two main types of interrupts: vectored and polled. Which type is the above interrupt? Explain.