Tutorial 1: Programming Model 1

Introduction

Objectives

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

- Use the CPU simulator to create basic CPU instructions
- Use the simulator to execute the basic CPU instructions
- Create instructions to move data to registers, compare values in registers, push data to the stack, pop data from the stack, jump to address locations, add values in registers.
- Explain the functions of special CPU registers such as the PC, SR and SP registers.

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

The programming model of computer architecture defines those low-level architectural components, which include the following

- Processor instruction set
- Registers
- Modes of addressing instructions and data
- Interrupts and exceptions

It also defines interaction between each of the above components. It is this low-level programming model which makes programmed computations possible.

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 a single window.

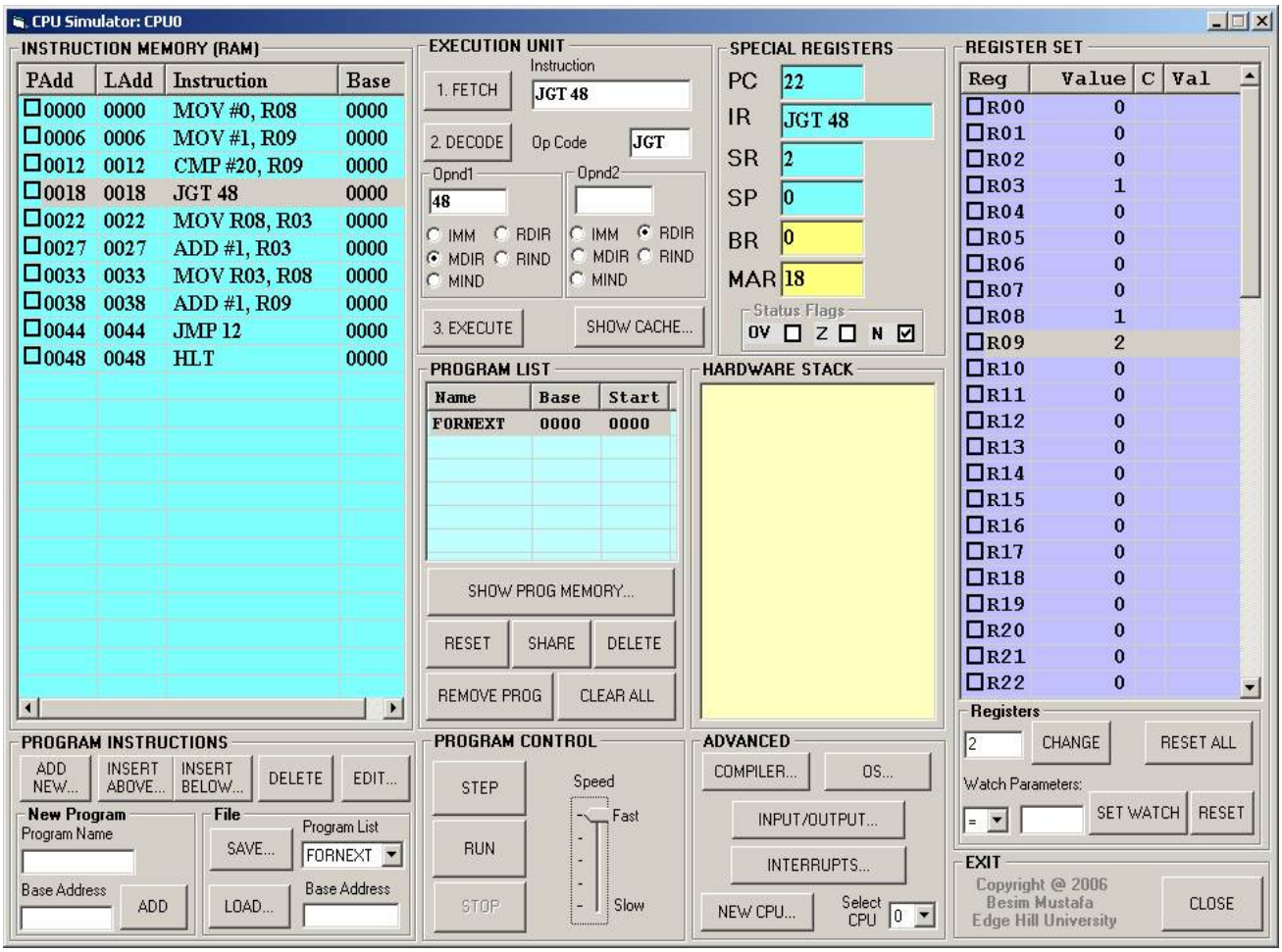


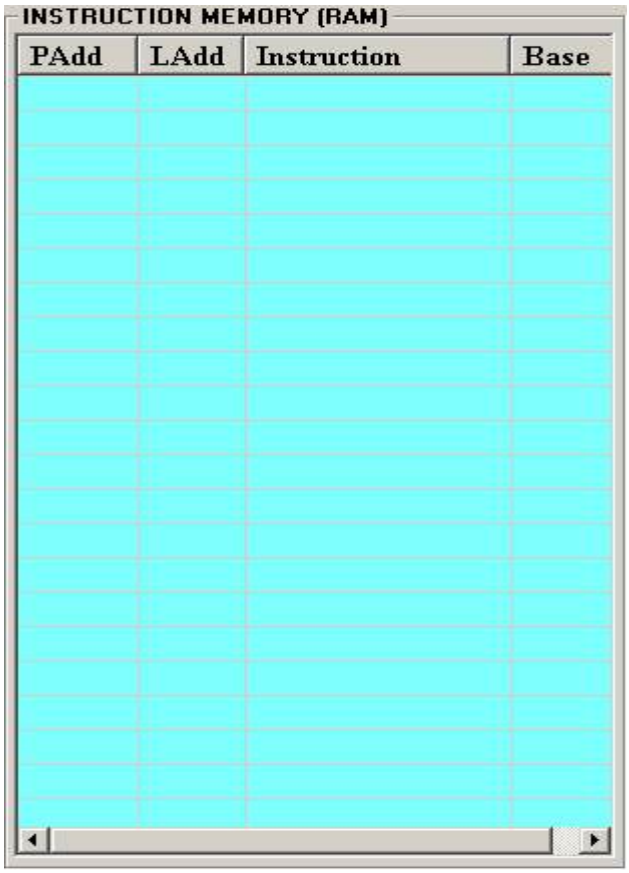
Image 1 - Main simulator window

The main window is composed of several views, which represent different functional parts of the simulated processor. These are

- Instruction memory
- Special registers
- Register set
- Hardware stack

The parts of the simulator relevant to this lab are described below.

Instruction memory view



This view contains the program instructions. The instructions are displayed as sequences of low-level instruction mnemonics (assembler-level format) and not as binary code. This is done for clarity and makes code more readable.

Each instruction has two addresses: the physical address (**PAdd**) and the logical address (**LAdd**). This view also displays the base address (**Base**) against each instruction. The sequence of instructions belonging to the same program will have the same base address.

Image2 - Instruction memory view

Special registers view

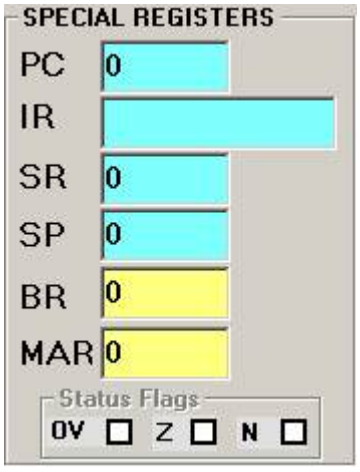


Image 3 - Special registers view

This view presents the set of registers, which have pre-defined specialist functions:

PC: **Program Counter** contains the address of the next instruction to be executed.
IR: **Instruction Register** contains the instruction currently being executed.
SR: **Status Register** contains information pertaining to the result of the last executed instruction.
SP: **Stack Pointer** register points to the value maintained at the top of the hardware stack (see below).
BR: **Base Register** contains current base address.
MAR: **Memory Address Register** contains the memory address currently being accessed.
Status bits: OV: Overflow; **Z**: Zero; **N**: Negative

Register set view

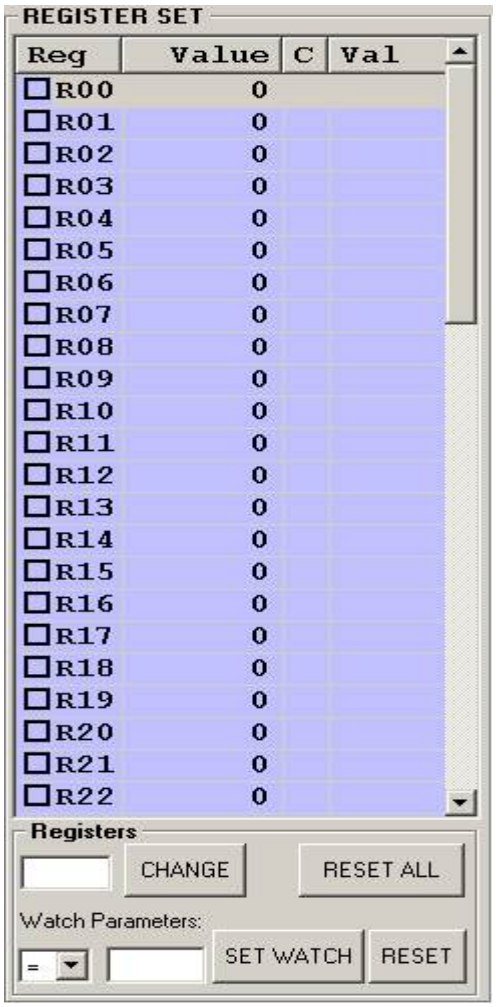


Image 4 - Register set view

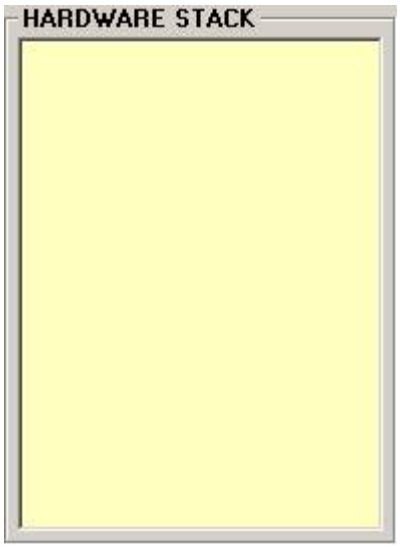
The register set view shows the contents of all the general-purpose registers, which are used to maintain temporary values as the program's instructions are executed.

This architecture supports from 8 to 64 registers. These registers are often used to hold values of a program's variables as defined in high-level languages.

Not all architectures have this many registers. Some have more (e.g. 128 register) and some others have less (e.g. 8 registers). In all cases, these registers serve similar purposes.

This view displays each register's name (**Reg**), its current value (**Value**) and some additional values, which are reserved for program debugging. It can also be used to reset the individual register values manually which is often useful for advanced debugging. To manually change a register's content, first select the register then enter the new value in the text box and click on the button **CHANGE**.

Hardware stack view



The hardware stack maintains temporary values as the instructions are executed. The stack is a LIFO (last-in-first-out) data structure. It is often used for efficient interrupt handling and sub-routine calls.

The instructions PSH (push) and POP are used to store values on top of stack and pop values from top of stack respectively.

Image 5 - Hardware stack view

Lab Exercises - Investigate and Explore

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

First we need to place some instructions in the **Instruction Memory View** (i.e. representing the RAM in the real machine) before executing any instructions. How are instructions placed in the Instruction Memory View?

Follow the following steps for this

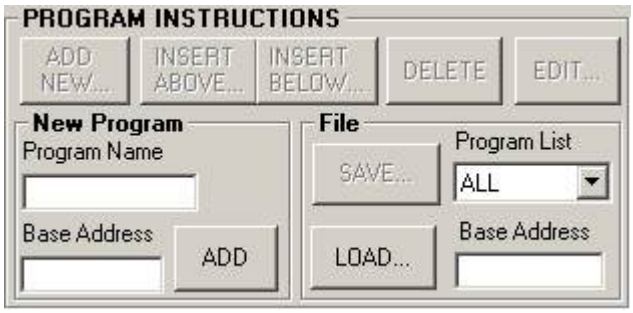
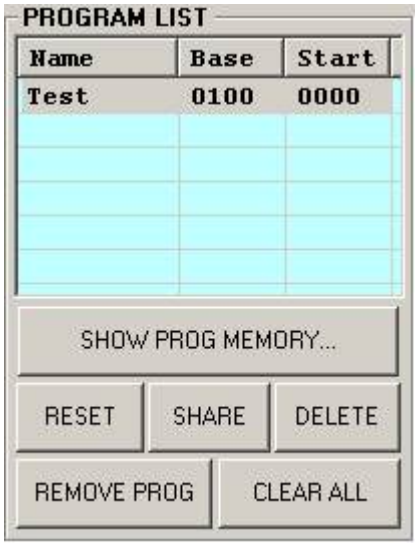


Image 6 - Program Instructions View

In the **Program Instructions View**, first enter a **Program Name**, and then enter a **Base Address** (this can be any number, but for this exercise use 100). Click on the **ADD** button. A new program name will be entered in the **Program List View** shown below. Use the **SAVE…** / **LOAD...** buttons to save instructions in a file and load the instructions from a file.



Use the **REMOVE PROG** button to remove the selected program from the list; use the **CLEAR ALL** button to remove all the programs from the list. Note that when a program is removed, its instructions are also removed from the **Instruction Memory View** too.

Image 7 - Program List View

IMPORTANT NOTE:

Before you carry on with the following tutorial exercises, first click on the **SHOW PIPELINE…** button in the **CPU Simulator** window and check the checkbox labelled **No instruction pipeline**. Close the window.

You are now ready to enter instructions into this CPU. You do this by clicking on the **ADD NEW…** button. This will display the **Instructions: CPU0** window. Use this window to enter the instructions. **Appendix** lists some of the instructions this simulator understands and gives a selection of examples.

Now, have a go at the following activities:

1. Create an instruction, which moves number 5 to register R00.
2. Execute the above instruction (simply double click on it in the Instruction Memory View).
3. Create an instruction, which moves number 8 to register R01.
4. Execute it.
5. Observe the contents of R00 and R01 in the Register Set View.
6. Create an instruction, which adds the contents of R00 and R01.
7. Execute it.
8. Observe where the result is put.
9. Create an instruction, which pushes the above result to the top of the hardware stack, and then execute it.
10. Create an instruction to push number -2 on top of the stack and execute it.
11. Observe the value in the SP register (Special Registers View).
12. Create an instruction to compare the values in registers R00 and R01.
13. Execute it.
14. Observe the value in the SR register.
15. Observe the status of the OV/Z/N bits of the status register.
16. Create an instruction to unconditionally jump to the first instruction.
17. Execute it.
18. Observe the value in the PC register. What is it pointing to?
19. Observe the values in the PAdd and LAdd columns. What do these values indicate? Are they different?

20. What is the difference between the LAdd value of the first instruction and the LAdd value of the second instruction? What does this value indicate?
21. Create an instruction to pop the value on top of the hardware stack into register R02.
22. Execute it.
23. Observe the value in the SP register.
24. Create an instruction to pop the value on top of the hardware stack into register R03.
25. Execute it.
26. Observe the value in the SP register.
27. Execute the last instruction again. What happened? Explain.
28. Create a compare instruction, which compares values in registers R04 and R05.
29. Manually insert two equal values in registers R04 and R05.
30. Execute the compare instruction in step 28 above.
31. Which of the status flags OV/Z/N is set? Why?
32. Manually insert a value in register R05 greater than that in register R04.
33. Execute the compare instruction in step 28 above.
34. Which of the status flags OV/Z/N is set? Why?
35. Manually insert a value in register R04 greater than that in register R05.
36. Execute the compare instruction in step 28 above.
37. Which of the status flags OV/Z/N is set? Why?
38. Create an instruction, which will jump to the first instruction if the values in registers R04 and R05 are equal.
39. Test this instruction by first putting appropriate values in registers R04 and R05 and then first executing the compare instruction and then executing the jump instruction.
40. Save the instructions in the Instruction Memory View in a file by clicking on the **SAVE…** button.

The above exercises are intended to help you understand some basic CPU instructions, which can be found in most modern CPU architectures. As often is the case, there is more to it than the above basic instructions.

Appendix - Simulator Instruction Sub-set

Instruction	Description
Data transfer instructions	
MOV	Move data to register; move register to register e.g. **MOV #2, R01** moves number 2 into register R01 **MOV R01, R03** moves contents of register R01 into register R03
LDB	Load a byte from memory to register
LDW	Load a word (2 bytes) from memory to register
STB	Store a byte from register to memory
STW	Store a word (2 bytes) from register to memory
PSH	Push data to top of hardware stack (TOS); push register to TOS e.g. **PSH #6** pushes number 6 on top of the stack **PSH R03** pushes the contents of register R03 on top of the stack
POP	Pop data from top of hardware stack to register e.g. **POP R05** pops contents of top of stack into register R05
Arithmetic instructions	
ADD	Add number to register; add register to register e.g. **ADD #3, R02** adds number 3 to contents of register R02 and stores the result in register R02. **ADD R00, R01** adds contents of register R00 to contents of register R01 and stores the result in register R01.
SUB	Subtract number from register; subtract register from register
MUL	Multiply number with register; multiply register with register
DIV	Divide number with register; divide register with register
Control transfer instructions	
JMP	Jump to instruction address unconditionally e.g. **JMP 100** unconditionally jumps to address location 100

JLT	Jump to instruction address if less than (after last comparison)
JGT	Jump to instruction address if greater than (after last comparison)
JEQ	Jump to instruction address if equal (after last comparison) e.g. **JEQ 200** jumps to address location 200 if the previous comparison instruction result indicates that the two numbers are equal.
JNE	Jump to instruction address if not equal (after last comparison)
CAL	Jump to subroutine address
RET	Return from subroutine
SWI	Software interrupt (used to request OS help)
HLT	Halt simulation
Comparison instruction	
CMP	Compare number with register; compare register with register e.g. **CMP #5, R02** compare number 5 with the contents of register R02 **CMP R01, R03** compare the contents of registers R01 and R03 Note: If R01 = R03 then the status flag Z will be set If R03 > R01 then non of the status flags will be set If R01 > R03 then the status flag N will be set
Input, output instructions	
IN	Get input data (if available) from an external IO device
OUT	Output data to an external IO device