Algorithms and Computational Thinking Autumn 2016

Thursday, 22th September 2016

EduMIPS64 - Guide

Introduction and basic information

Basically, the programs written in higher level languages (java, C (debatable, but some people still believe C is high level programming language), python etc.) when compiled, are translated to simple assembly level instructions that are then executed by the processor. Assemble language also known as machine language can be executed faster as compared to high level languages as they do not need to be compiled. Sometimes, machine language programs are inserted in between high level program to make some part highly optimized typically in mission critical projects such as space shuttle launches where even few micro seconds could make a difference. Therefore, it is important to get an idea how they are written and executed. The simulator EduMIPS64 was designed to execute small programs exactly as the processor would have executed it. We will give you a basic introduction to the simulator with simple examples and make you capable to play around with the simulator.

Every program written in assembly has two sections the *data* section and the *code* section. The data section contains commands that specify what should be filled in the memory before the program execution starts. While, code section contains commands that specify how the memory must be filled when the program will start.

How do I write an instruction in the .data field ?

label : .datatype value

A label can be any alphabet to keep it simple, its just used to different alphabet for different values you want to put in the data field. A simple program to load a number in the memory is presented below.

```
; My first program (this is how you write a comment in assembly language)
.data
a : .word 15
b : .word 70 ; loading a second word
.halt ; every code ends with a .halt, to tell the process that the code is terminated
```

A datatype can either be a byte, half word, word or double word. Each one reserves different number of bits in the memory for the declared value. But for simplicity lets just stick to .word for now. .word reserves 32 bits for the declared value, its already quite a lot.

So now, that you know how to insert values in the memory, which will be used in your code, lets see how to write the code section.

The beginning of a code section starts with a .text, to tell the assembler the start of the code section. In the processor, there are spaces called *registers* that are accessed and operations are performed on them. So basically, assemble language programming is just manipulating the register values. So lets learn some instructions on how to manipulate them.

The first thing to be done is to move the values loaded in the memory to the registers, so that we can perform some operations on them. We can also directly perform operations on the memory values, but to access the memory it takes time as compared to accessing the registers. Think of it, as your memory inside your computer and the external flash drive you connect. So, to the processor, it is easier and faster to access register contents. So how do we do it ? We use the instruction to move the value from main memory to the register. The instruction is called *ld*.

The complete instruction is *ld rt, offset(base)*

Loads the content of the memory cell at address specified by offset and base in register *rt*, treating it as an unsigned byte. To make it simple, offset is the label associated to the value you want to move to the register. There are a total of 32 registers in 64 bit architecture based processors. So you can use all of them, they are named from r0, r1...r32. So *rt* can be any one of them. The base on the other hand will be always r0 in our case. (If you have doubts regarding what exactly the base is you can ask us during the lab sessions). But for simplicity lets just assume that in our case it will be always r0.

```
; Code to load a word in memory and move it to register r4 .data
A : .word 10
.text
main :; you need to give a label to the code section, lets call it main for now,
but it can be anything.
ld r4, A(r0) ; move the value A to r4
.halt
```

Using the Simulator

1. Open the simulator by double clicking on the edumips64-1.2.2.jar file ;
2. Open a file in any of your favorite text editor (except emacs ;)). We have installed *Atom*, so you can use it ;
3. Type in the code. Save the file as anything ;
4. Open the file in the simulator and run it.