CS204 - Computer Architecture
Practice Lab - 2
Date: 27th Jan 2025. Not Graded!
Deadline: 3rd Feb 2025. Complete all the tasks at the earliest!
Working with RISC - V using Venus

We will continue working with RISC-V ISA using *Venus* simulator. Over the last one week you have been introduced to many RISC - V instructions in the class and tutorial. In this lab, try out as many different instructions as you can, to perform the given tasks. Use the RISC-V Reference Data sheet for any quick look up and the class slides to know the role of each instruction. Below tasks are not exhaustive but are only indicative for your practice.

Task 1: Using RISC-V Data Transfer Instructions
Use a `store` instruction to store `0xdeadbeef` into the memory. The simulator can display memory. Click on the Memory field, scroll down and Jump to Data. The simulator choses to start the data segment at `0x1000000`. Now write into that location the data (`0xdeadbeef`). How do you get that address into a register in first place? Remember immediate values? After storing that value into the main memory also read it back into another register.

Task 2: Assembler Directives

Beside instructions in assembly format, an assembler also accepts so-called *assembler directives*. The beginning of code segment is usually marked with `.text`. You can initialize data in the data segment with `.data`. Each assembly instruction can start with a label such as `main:` or `loop:`. As you know, this label can then be used as destination in control instructions. Similarly data can be also addressed by using a label. See below some examples:

```
   .data
label1:
.asciiz "Hello"

.text
la a1, label1
label2:  li, x3, 123
# ...  more code
```

'la' is another pseudo instruction that Venus supports.
Now add a `main:` label to the start of your code (i.e., `.text` segment) and add following instruction at the end of your program:

```
j main
```

What happens when you step through your code? What happens when you press `Run`? Check the RISC-V reference data sheet to know more about the pseudo instruction '*j*'.

Task 3: RISC-V immediate values for ALU operations are 12-bit wide. Try to use larger constants in your program (you can also use the `0xabcd` notion for hexadecimal values).

(a) What happens when the constant does not fit into sign extended 12 bits?
(b) Lookup the `lui` instruction in RISC-V Reference Data sheet. Try using it.

Task 4: Always remember that when you start writing a code to perform any task, the memory is empty. One of the first tasks is to *fill* the memory with the values that you would be using in the code. In the previous lab, you were introduced to two assembler directives, namely, `.text` and `.data` to mark the code/text segment and data segment, respectively. Henceforth, write all your assembly codes by utilizing these directives.

Consider the following task at hand - We need to add two elements, say, 10 and 20. Even though we can use *li* instruction to load these values into the registers and add them (Task 2 in the Lab 1.2), it would be helpful to understand if we can separate the data, and the code we write to operate on the data. This practice will help us to visualize and organize the assembly code better. It will come to your aid when you begin writing huge codes and will be of great help during debugging.

Enter the following code into the Editor pane and switch to the Simulator pane.

```
# Sample code
.data
var1:  .word 10
var2:  .word 20
.text
lw x1, var1
lw x2, var2
add x3, x2, x1
```

var1 and *var2* are labels we are using for our convenience. `.word` is another assembler directive that will help us reserve one word for the number given.
1. In the Simulator pane, on the right, click on Memory and select Data in Jump to. Note the locations where var1 and var2 are stored.
2. Replace one of the `.word` with `.byte` and observe both the Data in Memory tab, and the Registers at the end of execution. Increase the value and observe the difference in the result of the *add* operation.
3. Replace the `.data` segment with below lines

```
.data
var1:  .byte 10
var2:  .byte 20
var3:  .word 0xffffffff
```

Now, in the Memory, note the addresses the variables *var1, var2, var3* are stored in. Further, check the addresses where the instructions (after `.text`) are stored. This is one of the examples of relaxing 'word-alignment' for the data segment. Did you realize that?
For the curious ones: You can check for all the assembler directives supported by RISC-V here *https* : *//rv*8*.io/asm.html*

For the rest of the tasks, preload your memory with the data you would like to use, label the data and use the labels in your code segment.

Task 5: Realizing a 'if-else' statement in RISC-V.
In class, we have studied the following C code

```
# Simple if-else in C
if(i == j)
   f = g + h;
else
   f = g - h;
```

Write the corresponding RISC-V code using
1. *beq* instruction
2. *bne* instruction.

Also, observe the 'Original code' and 'Basic code' parts in the Simulator pane. You can notice that the instruction labels you have used are replaced with values. Convince yourself that the values are appropriate replacement of the labels.

Task 6: In general, when we use `gcc` to compile a C (C++) program, it does both the tasks of Compilation and Assemble. You can stop this process just to get the compilation done, by using `-S` flag. It will generate a file (with ".s" extension) that contains the assembly code of your HLL code.
Write a simple "hello world" program in C, generate the assembly file (mostly it will be x86 ISA, check the ISA of your lab system!) and try figuring out the code and data segment, various labels used, assembler directives, etc.,
For the curious ones: A comprehensive list of available options for gcc are provided here: *https* : *//gcc.gnu.org/onlinedocs/gcc*-4.3.2/*gcc/Overall-Options.html#Overall-Options*

Task 7: Once you are finished with *all* the above tasks, upload a single text file (name it "YourName_YourRollNo_learnings_lab2.txt") documenting your learnings and interesting aspects that you have noticed today. Write point-wise statements and not paragraphs. Upload the file by Friday, 3rd Feb 2025, 11.55AM.