Introductory Computer Organization
Lab#2

In this lab, you will learn how to use MIPS assembly instructions to:

1. implement FOR and WHILE loop constructs
2. read/write from/to memory

Download the lab files and implement two functions “strlen” and “valid_id”. Please take a look at comments in the provided source code for more information.

1. Implementing loop constructs in MIPS assembly:

1.1. WHILE loop:

The most general form of loop is WHILE loop which takes the following form in C/C++:


```
while (condition) {
     <<loop body statements>>
}
```


There are two important points to note about a WHILE loop. First, the test for termination appears at the beginning of the loop. Second, as a direct consequence of the position of the termination test, the body of the loop may never execute.

A WHILE loop can be converted to assembly using the following pseudo-code template:

```
BeginningOfWhile:
If (condition) {
     <<loop body statements>>
     j BeginningOfWhile
}
EndOfWhile:
```

Therefore, you can use the technique from earlier lab to convert IF statements to assembly language along with a single JMP instruction to produce a WHILE loop. An example of such translation is shown below:

C code	MIPS Assembly code

`while ( c > 0 ) {`
`  a = b + 1;`
`  c--;`
`}`
	`lw $t1, a`
	`lw $t2, b`
	`lw $t3, c`
	
	`BeginningOfWhile:`
	`blez $t3, EndOfWhile`
	`addi $t1, $t2, 1`
	`subi $t3`
	`j BeginningOfWhile`
	
	`EndOfWhile:`
	`sw $t1, a`
	`sw $t3, c`

1.2. FOR loop:

A FOR loop is essentially a WHILE Loop with an initial state, a condition, and an iterative instruction. For instance, the following generic FOR Loop in C/C++:

```
For (initialization; condition; update) {
     <<loop body>>
}
```

can be converted to the following pseudo-code using WHILE loop:

```
<<initialization>>
while(condition) {
     <<loop body statements>>
     <<update>>
```

```
}
```

We already know how to translate a WHILE loop to x86 assembly, therefore we can translate any FOR loop using the above template.

2. MIPS Memory Addressing Modes:

The memory addressing mode determines, for an instruction that accesses a memory location, how the address for the memory location is specified. Two modes that we use to access data in MIPS assembly are:

1) **register addressing mode**: this is the simplest form to access memory. In this form, memory address is stored in a register.
2) **base addressing mode**: address is stored in a register, however it can be adjusted by an immediate value.

To obtain the address of data in memory we use the "la" instruction. You can access memories in byte granularity with "lb, sb" or word granularity with "sw, lw". In this lab, since data elements are character, we use lb and sb.

Mode	Example	Effect
Register addressing	`lw $t2, ($t3)`	`$t2 = Mem[$t3]`
	`sw $t3, ($t4)`	`Mem[$t4] = $t3`
Base addressing	`lw $t2, 32($t3)`	`$t2 = Mem[$t3 + 32]`
	`Sw $t3, 16($t4)`	`Mem[$t4 + 16] = $t3`