CS 21 Laboratory 3: Hello! The Data Field and Load-Store Operations

University of the Philippines - Diliman
College of Engineering
Department of Computer Science

1 Data field

1.1 .ascii and .byte directive

We could explicitly populate our data memory with data via the data field. For example, to place the characters comprising the word "Hello" in the memory, we could use the **.ascii** directive, as shown in Code Block 1.

Code Block 1 Using the .ascii and .byte directive

```
.data
hello: .ascii "H" # The letter "H"
.ascii "e" # The letter "e"
.ascii "l" # The letter "l"
.ascii "l" # The letter "l"
.ascii "o" # The letter "o"
.ascii "\n" # A newline.
.byte 0 # zero
.byte 0 # zero
```

Take note that what **.ascii** does is to convert our character into its ASCII equivalent number, and store that number in the memory. An ASCII code for a character takes up 8 bits or one byte. A word in the MIPS memory is 4 bytes or 32 bits. In storing the word "Hello", how many memory words did we use? What is the directive **.byte** for? Why did we have to use it twice?

Also, try using

```
.asciiz "Hello\n"
```

What does this do?

2 print_string syscall

Since the characters are stored in binary/hex format, they are not readily human-readable, even if we could see the memory values in the lowest window. To convert them back to characters and print them in the console, we use the print_string syscall, demonstrated in Code Block 2 (we assume that we keep the code from Code Block 1.

Take note that in using **la**, we are loading the actual address value "aliased" by the label. In using print_string, such value must always be loaded to register a0.

Code Block 2 The print_string syscall

```
la $a0, hello # load the addr of hello into $a0.
li $v0, 4 # 4 is the print_string syscall.
syscall # do the syscall.
```

3 Byte addressability

Memory in the MIPS architecture is byte-addressable. That is, the smallest unit of memory that we could address is 8 bits or a byte.

3.1 Loading bytes and halfwords

We could load bytes and halfwords using the instructions **lb**, **lh**, **lbu** and **lhu**.

For example, assuming that we want to load register t4 with the third byte in the word labelled "Hello", we use Code Block 3. We DO NOT change the address pointed to by hello. We create a copy of that address (using **la**), and then increment it by 2. The effect of this incrementation is that the "pointer" now moves "two steps"(actually, two bytes) to the left.

Code Block 3 Example of load byte unsigned usage

```
la $t3, hello # load address of hello to t3
addi $t3, $t3, 2 # increment t3
lbu $t4, ($t3)
```

4 Machine Exercise

Create an assembly language program that does the following.

- takes in as input(via the console) a 6-lettered word, all in uppercase
- prints as output(via the console) the same word, but this time every other letter should be in lowercase.
- for example if the input is "THANKS" the output must be "ThAnKs"

Save the program under the filename **cs21me3.asm**.
Call my attention upon being able to complete the ME for checking.

5 Learning Checklist

In this session you should have learned...

- how to place data in memory using the .data directive
- how to address, load, and store words, halfwords, and bytes.