

國立清華大學資訊工程學系
10610 CS 410000 計算機結構
Homework 2

Deadline: 2017.10.10 23:59

There are two parts in this homework.

PART I. (Load, Store, Add, Sub)

Please load the data 0(\$gp) as A, 4(\$gp) as B, and do the following calculations.

C = A + B, store C to 8(\$gp).

D = B - A, store D to 12(\$gp)

Hint

We will give a template called **arch_hw2_p1_template.asm**, just open it using Mars4_5.jar, write your code within the **#####** block in the file (i.e., line 36~41), but **DO NOT** modify the code elsewhere. Please refer to the following figure.

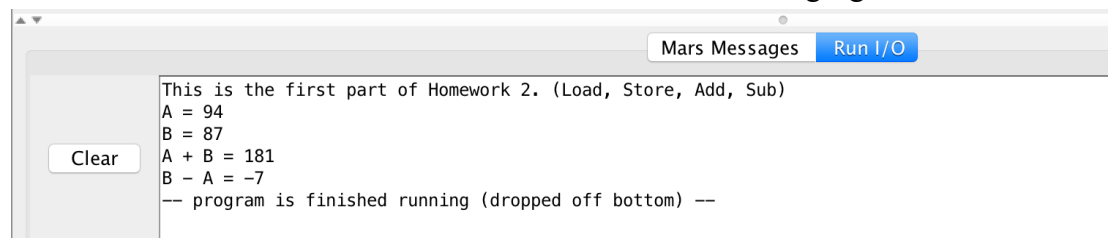
```
29 # The following block helps you practice with the R-type instructions,
30 # including ADD and SUB, and also LOAD and STORE.
31 # Here, please read data from 0($gp) and 4($gp),
32 # store 0($gp)+4($gp) to 8($gp), and
33 # store 4($gp)-0($gp) to 12($gp)
34 #####
35 # @@@ write the code here
36
37
38
39
40
41
42 #####
```

After you write your code, save it.

Next, press “F3” to assemble the code. (**Make sure there is no error!**)

Next, press “F5” to run.

The “Run I/O” screen should show the result like the following figure.



PART II. (Branch Loop, System call, Four Arithmetic Operations)

Please convert the following C-like code to MIPS assembly code. Write a new assembly file for this part.

Description: Enter a 2-digit number, and test whether it is a divine number. (Note that all arithmetic operations are integer-type)

```
for (;;) {
    printf("Please enter a 2-digit number(10~99): ");
    scanf("%d", &$t0);
    if ($t0 == 0) break;
    else if ($t0 < 10 || $t0 > 99) {
        printf("$t0 = %d is NOT a 2-digit number!\n", $t0);
        continue;
    }
    else {
        $t1 = $t0 / 10;
        $t2 = $t0 % 10;
        $t3 = $t1 + $t2;
        $t4 = $t1 * $t2;
        $t5 = $t4 / 4;
        while ($t5 --> 0)    $t3 <= 1;
        ($t3 + $t4 == $t0) ?
            printf("$t0 = %d is a divine number!\n", $t0):
            printf("$t0 = %d is NOT a divine number!\n", $t0);
    }
}
```

Hint

- You can refer to the template in Part I or Appendix to learn how to do `printf` and `scanf` in MIPS.
- You can refer to the following MIPS example to learn how to find the integer quotient and the remainder of any two numbers.

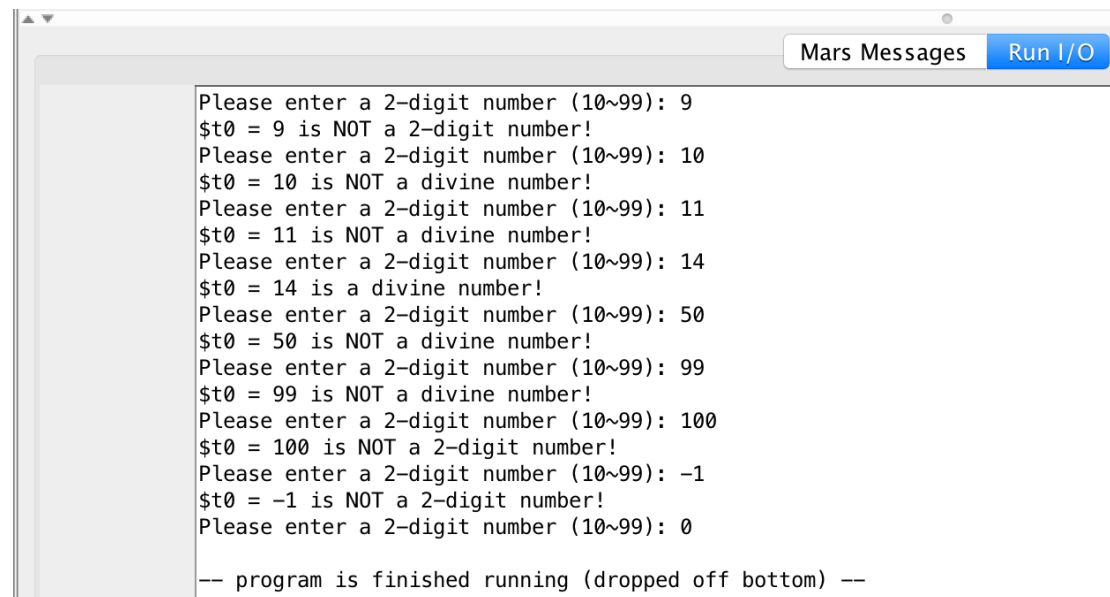
(Source: <http://logos.cs.uic.edu/366/notes/mips%20quick%20tutorial.htm>)

```
div  $t5, $t6    # Lo = $t5 / $t6 (integer quotient)
                # Hi = $t5 % $t6 (remainder)
mfhi $t0         # move quantity in special register Hi to $t0: $t0 = Hi
mflo $t1         # move quantity in special register Lo to $t1: $t1 = Lo
                # used to get at result of product or quotient
```

A simple test flow is like the following “Run I/O” screenshot.

Consecutive keystrokes (delimiter: ","):

9, enter, 10, enter, 11, enter, 14, enter, 50, enter, 99, enter, 100, enter, -1, enter, 0, enter



```
Please enter a 2-digit number (10~99): 9
$t0 = 9 is NOT a 2-digit number!
Please enter a 2-digit number (10~99): 10
$t0 = 10 is NOT a divine number!
Please enter a 2-digit number (10~99): 11
$t0 = 11 is NOT a divine number!
Please enter a 2-digit number (10~99): 14
$t0 = 14 is a divine number!
Please enter a 2-digit number (10~99): 50
$t0 = 50 is NOT a divine number!
Please enter a 2-digit number (10~99): 99
$t0 = 99 is NOT a divine number!
Please enter a 2-digit number (10~99): 100
$t0 = 100 is NOT a 2-digit number!
Please enter a 2-digit number (10~99): -1
$t0 = -1 is NOT a 2-digit number!
Please enter a 2-digit number (10~99): 0

-- program is finished running (dropped off bottom) --
```

Hint

- We will give the C code called **arch_hw2_p2.c** for reference. We will also give a MIPS template called **arch_hw2_p2_template.asm**. We strongly recommend you do this part by yourself, or you can refer to the template if you need some help.
- 14 is one divine number but is not the only one, and TA will use another divine number to test if your program is correct.

Submission (Two assembly programs)

Please name your assembly program with your student ID; for example, **arch_hw2_p1_105062901.asm** & **arch_hw2_p2_105062901.asm**, and upload these 2 files onto iLMS. (<http://lms.nthu.edu.tw/course/30643>)

Grading Criteria

Correctness: 80%

Comments in your code: 10%

Output format: 10%

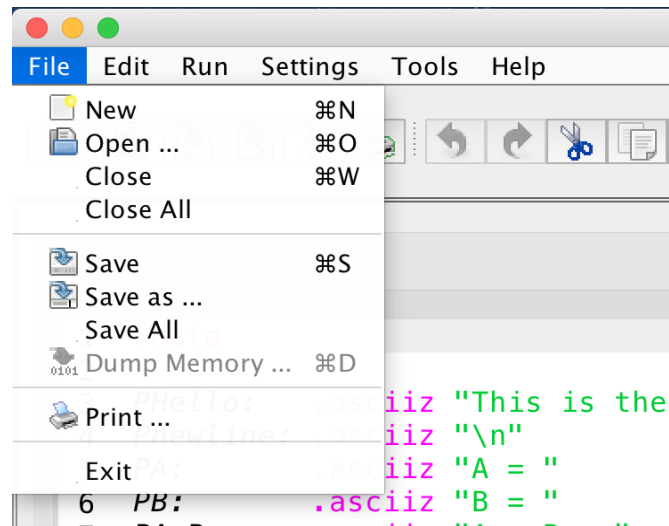
MARS (MIPS Assembler and Runtime Simulator)

- MARS can assemble and simulate the execution of MIPS assembly language programs. Please refer to the following URL to download Mars4_5.jar:
<http://courses.missouristate.edu/kenvollmar/mars/download.htm>
- MARS is developed with Java language, and it requires JRE (Java Runtime

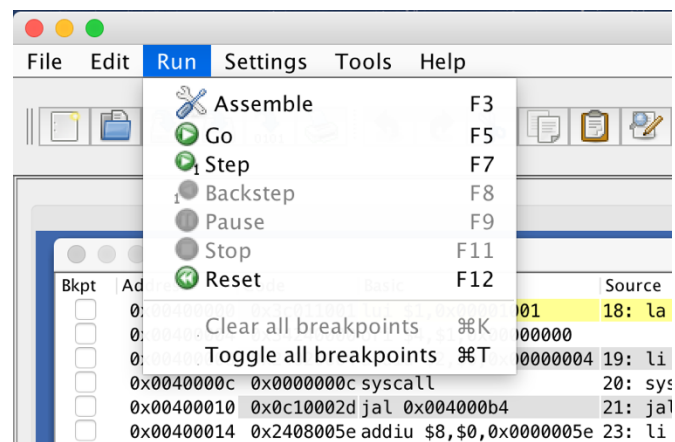
Environment) installed on your computer. Please refer to the following URL to download JRE 9:

<http://www.oracle.com/technetwork/java/javase/downloads/jre9-downloads-3848532.html>

3. Usage of MARS:



(a) New, Open, Save and Close



(b) Assemble and then Go (Run)

Appendix

(Source: <http://students.cs.tamu.edu/tanzir/csce350/reference/syscalls.html>)

MIPS system calls

(from SPIM S20: A MIPS R2000 Simulator, James J. Larus, University of Wisconsin–Madison)

SPIM provides a small set of operating-system-like services through the MIPS system call (syscall) instruction. To request a service, a program loads the system call code (see Table below) into register \$v0 and the arguments into registers \$a0, ..., \$a3 (or \$f12 for floating point values). System calls that return values put their result in register \$v0 (or \$f0 for floating point results).

Service	System Call Code	Arguments	Result
print integer	1	\$a0 = value	(none)
print float	2	\$f12 = float value	(none)
print double	3	\$f12 = double value	(none)
print string	4	\$a0 = address of string	(none)
read integer	5	(none)	\$v0 = value read
read float	6	(none)	\$f0 = value read
read double	7	(none)	\$f0 = value read
read string	8	\$a0 = address where string to be stored \$a1 = number of characters to read + 1	(none)
memory allocation	9	\$a0 = number of bytes of storage desired	\$v0 = address of block
exit (end of program)	10	(none)	(none)
print character	11	\$a0 = integer	(none)
read character	12	(none)	char in \$v0

For example, to print "the answer = 5", use the commands:

```
.data
str: .asciiz "the answer = "
.text
    li $v0, 4      # $system call code for print_str
    la $a0, str     # $address of string to print
    syscall         # print the string

    li $v0, 1      # $system call code for print_int
    li $a0, 5      # $integer to print
    syscall         # print it
```

- **print int** passes an integer and prints it on the console.
- **print float** prints a single floating point number.
- **print double** prints a double precision number.
- **print string** passes a pointer to a null-terminated string
- **read int**, **read float**, and **read double** read an entire line of input up to and including a newline.

- **read string** has the same semantics as the Unix library routine `fgets`. It reads up to $n - 1$ characters into a buffer and terminates the string with a null byte. If there are fewer characters on the current line, it reads through the newline and again null-terminates the string.
- **sbrk** returns a pointer to a block of memory containing n additional bytes.
- **exit** stops a program from running.