國立清華大學資訊工程學系

10910 CS 410000 計算機結構

Homework 2

Deadline: 2020.10.19 (23:59)

There are two parts in this homework.

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

Please load the data 12($gp) as A, 0($gp) as B, and 8($gp) as C, and do the following calculations.

`D = C - B + A`, store `D` to `4($gp)`.

`E = A + A + C`, store `E` 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.

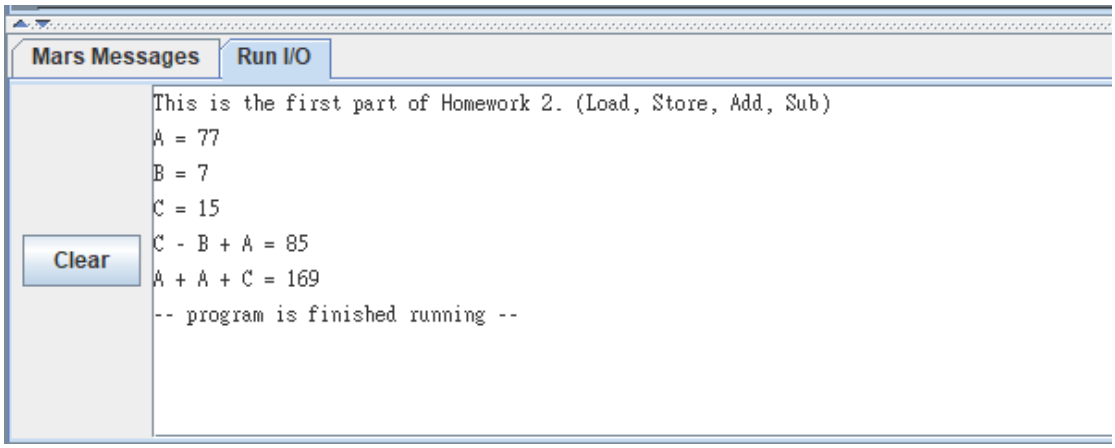
```
33  # The following block helps you practice with the R-type instructions,
34  # including ADD and SUB, and also LOAD and STORE.
35  # Here, please read data from 12($gp), 0($gp) and 8($gp),
36  # store C - B + A to 4($gp), and
37  # store A + A + C to 12($gp)
38  ##################################################
39  # @@@ write the code here
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55  ##################################################
```

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, Arithmetic Operations)

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

Description: Please design a “Toy MIPS Bomb” that will generate a random number (from zero to nine), and you can enter your lucky number into this Bomb (for only three times). If your lucky number equals the random number, you can save the world.

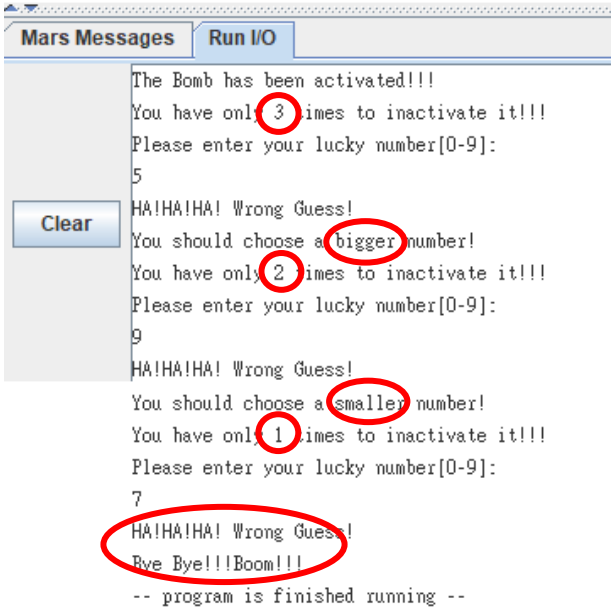
```c
#include<stdio.h>

int main()
{
    int t0;
    int bingo,test1,test2;
    int counter = 3;

    printf("The Bomb has been activated!!!\n");
    srand(time(NULL));
    bingo = (rand()%9);

    while(counter > 0)
    {
        printf("You have only %d times to inactivate it!!!\n", counter);
        printf("Please enter your lucky number[0-9]:\n");
        scanf("%d", &t0);

        if(t0 < bingo)
        {
            if(counter>1)
            {
                printf("HA!HA!HA! Wrong Guess!\n");
                printf("You should choose a bigger number!\n");
            }
            else
            {
                printf("HA!HA!HA! Wrong Guess!\n");
                printf("Bye Bye!!!Boom!!!\n");
            }
        }
        else if(t0 > bingo)
        {
            if(counter>1)
            {
                printf("HA!HA!HA! Wrong Guess!\n");
                printf("You should choose a smaller number!\n");
            }
            else
            {
                printf("HA!HA!HA! Wrong Guess!\n");
                printf("Bye Bye!!!Boom!!!\n");
            }
        }
        else
        {
            printf("Wow! You are very lucky! The boom has been inactive!\n");
            return 0;
        }
        counter = counter - 1;
    }
    return 0;
}
```

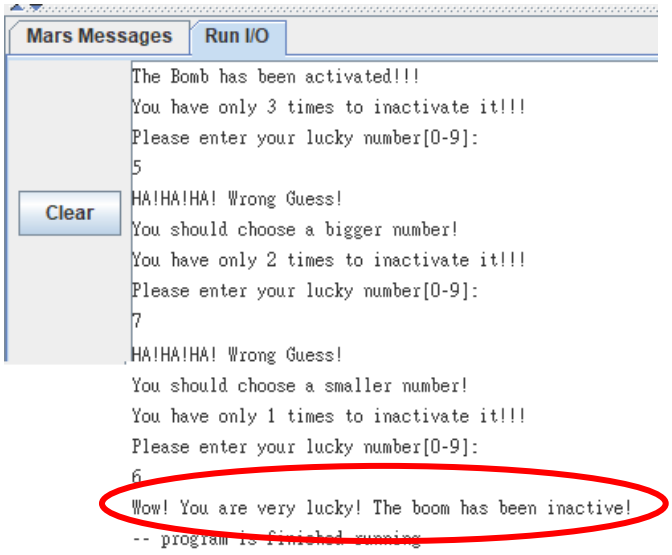
Hint

a. You can refer to the template in Part I or Appendix to learn how to do `printf` and `scanf` in MIPS.

b. Two references for finding the functionality of MIPS instruction.
(English: http://alumni.cs.ucr.edu/~vladimir/cs161/mips.html)
(中文: https://blog.xuite.net/tzeng015/twblog/113272086-MIPS+%E6%8C%87%E4%BB%A4%E9%9B%86)

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





Hint

a. 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.
b. TA will use other numbers 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_102062801.asm** & **arch_hw2_p2_102062801.asm**, and upload these 2 files onto iLMS. (https://lms.nthu.edu.tw/course/46255)

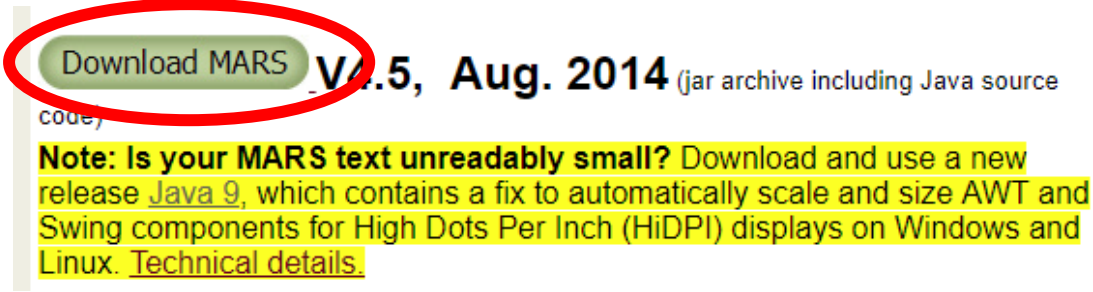
Grading Criteria

Correctness: 80%
Comments in your code: 10%
Output format: 10%

嚴格禁止抄襲，抄襲者與被抄襲者一律 0 分！

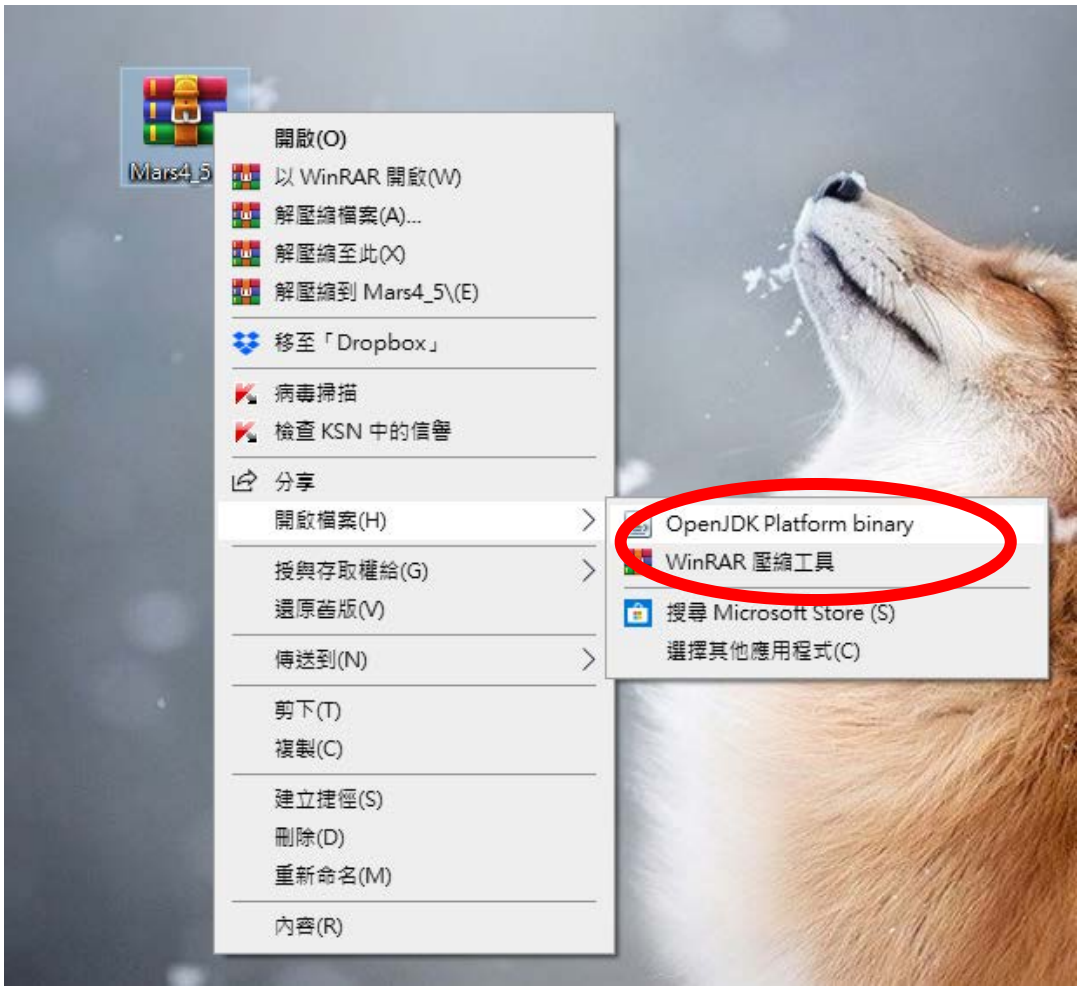
MARS (MIPS Assembler and Runtime Simulator)

1. 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

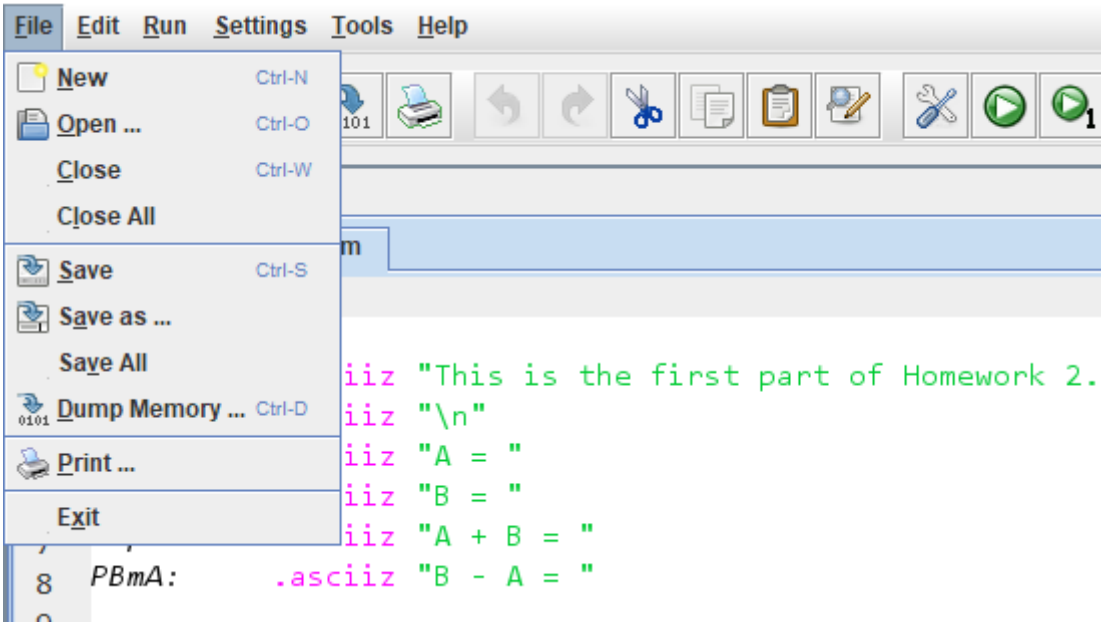


2. MARS is developed with Java language, and it requires JRE (Java Runtime Environment) installed on your computer. Please refer to the following URL to download JRE: https://www.oracle.com/java/technologies/javase-jdk15-downloads.html

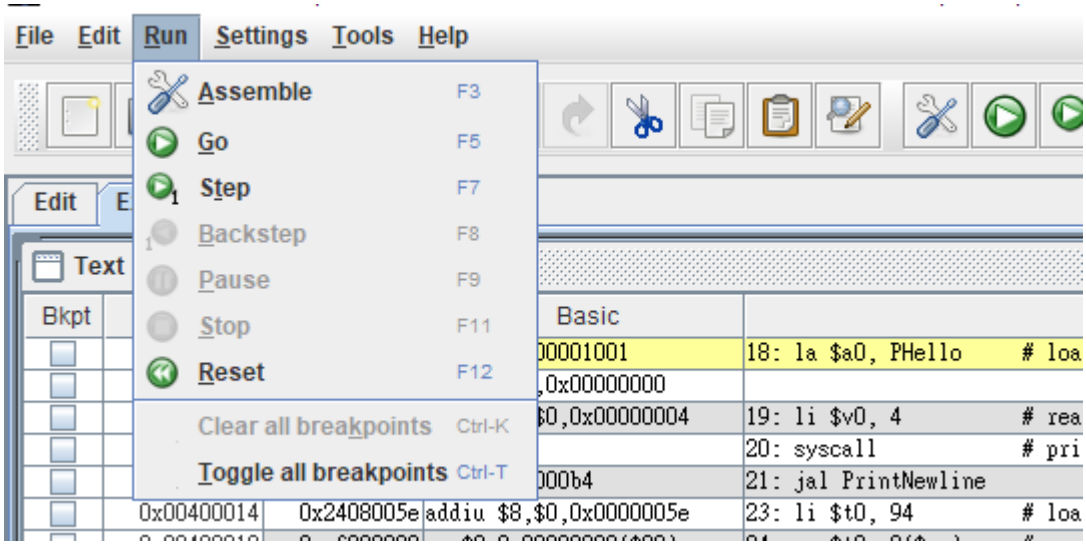
3. After you download the MARS, it is a ".jar" file. Please **DO NOT** decompress it. You can open the MARS by following method.



4. Usage of MARS:



(a) New, Open, Save and Close



(b) Assemble and then Go (Run)

P. S. Save your file, Assemble and Go

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.