Week 05 Weekly Test Sample Answers

Test Conditions

These questions must be completed under self-administered exam-like conditions. You must time the test yourself and ensure you comply with the conditions below.

- You may complete this test in CSE labs or elsewhere using your own machine.
- You may complete this test at any time before **Tue Oct 27 21:00:00 2020**.
- Weekly tests are designed to act like a past paper - to give you an idea of how well you are progressing in the course, and what you need to work on. Many of the questions in weekly tests are from past final exams.
- Once the first hour has finished, you must submit all questions you've worked on.
- You should then take note of how far you got, which parts you didn't understand.
- You may choose then to keep working and submit test question anytime up to Tue Oct 27 21:00:00 2020
- However the maximum mark for any question you submit after the first hour will be 50%

You may access this **language documentation** while attempting this test:

- C quick reference
- MIPS quick reference

You may also access manual entries (the `man` command).

Any violation of the test conditions will results in a mark of zero for the entire weekly test component.

Set up for the test by creating a new directory called `test05`, changing to this directory, and fetching the provided code by running these commands:

```
$ mkdir test05
$ cd test05
$ 1521 fetch test05
```

Or, if you're not working on CSE, you can download the provided code as a zip file or a tar file.

WEEKLY TEST QUESTION:

MIPS Minimum

In the files for this lab, you have been given `min.s`, a MIPS assembler program that reads 2 numbers and then prints 42.

Add code to `min.s` to make it equivalent to this C program:

```c
// print the minimum of two integers
#include <stdio.h>

int main(void) {
    int x, y;

    scanf("%d", &x);
    scanf("%d", &y);

    if (x < y) {
        printf("%d\n", x);
    } else {
        printf("%d\n", y);
    }

    return 0;
}
```

In other words it should read 2 numbers and print the smaller number.

For example:

```
$ 1521 spim -f min.s
5
8
5
$ 1521 spim -f min.s
118
26
26
$ 1521 spim -f min.s
42
42
42
```

When you think your program is working you can `autotest` to run some simple automated tests:

```
$ 1521 autotest min
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs1521 test05_min min.s
```

Sample solution for `min.s`

```
#  print the minimum of two integers
# x in $t0, y in $t1
main:
    li   $v0, 5         #   scanf("%d", &x);
    syscall             #
    move $t0, $v0

    li   $v0, 5         #   scanf("%d", &y);
    syscall             #
    move $t1, $v0

    bge  $t0, $t1, else # if (x < y) {
    move $a0, $t0       #   printf("%d\n", x);
    li   $v0, 1
    syscall
    li   $a0, '\n'      #   printf("%c", '\n');
    li   $v0, 11
    syscall
    j    end

else:

    move $a0, $t1       #   printf("%d\n", y);
    li   $v0, 1
    syscall
    li   $a0, '\n'      #   printf("%c", '\n');
    li   $v0, 11
    syscall

end:

    li   $v0, 0         # return 0
    jr   $ra
```

Alternative solution for `min.s`

```
#  print the minimum of two integers - more concise solution
# x in $t0, y in $t1
main:
    li   $v0, 5         #   scanf("%d", &x);
    syscall             #
    move $a0, $v0

    li   $v0, 5         #   scanf("%d", &y);
    syscall             #

    bge  $v0, $a0, end
    move $a0, $v0
end:

    li   $v0, 1         #   printf("%d\n", y);
    syscall
    li   $a0, '\n'      #   printf("%c", '\n');
    li   $v0, 11
    syscall


    li   $v0, 0         # return 0
    jr   $ra
```

WEEKLY TEST QUESTION:

MIPS Counting (but not 13)

In the files for this lab, you have been given not13.s, a MIPS assembler program that reads 2 numbers and then prints 42.

Add code to `not13.s` to make it equivalent to this C program:

```
// print the integers between x and y except 13
#include <stdio.h>

int main(void) {
    int x, y;

    scanf("%d", &x);
    scanf("%d", &y);

    int i = x + 1;
    while (i < y) {
        if (i != 13) {
            printf("%d\n", i);
        }
        i = i + 1;
    }

    return 0;
}
```

In other words it should read 2 numbers and print the numbers between them, except it doesn't print 13.

For example:

```
$ 1521 spim -f not13.s
5
8
6
7
$ 1521 spim -f not13.s
10
15
11
12
14
$ 1521 spim -f not13.s
5
14
6
7
8
9
10
11
12
```

Assumptions/Limitations/Clarifications

You can assume the first number read is not greater than the second number.

When you think your program is working you can `autotest` to run some simple automated tests:

```
$ 1521 autotest not13
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs1521 test05_not13 not13.s
```

Sample solution for `not13.s`

```
#  print the integers between x and y except 13
# x in $t0, y in $t1, i in $t2
main:
    li   $v0, 5         #   scanf("%d", &x);
    syscall             #
    move $t0, $v0

    li   $v0, 5         #   scanf("%d", &y);
    syscall             #
    move $t1, $v0

    addi $t2, $t0, 1    # i = x + 1;

loop:
    bge  $t2, $t1, end  # while (i < y) {
    beq  $t2, 13, skip  #   if (i != 13) {
    move $a0, $t2       #     printf("%d\n", i);
    li   $v0, 1
    syscall
    li   $a0, '\n'       #     printf("%c", '\n');
    li   $v0, 11
    syscall
skip:                   #   }
    addi $t2, $t2, 1    #   i = i + 1;
    j    loop           # }

end:

    li   $v0, 0         # return 0
    jr   $ra
```

WEEKLY TEST QUESTION:

MIPS Squares

In the files for this lab, you have been given square.s, a MIPS assembler program that reads a number and then prints 42.

Add code to `square.s` to make it equivalent to this C program:

```c
// print a square of asterisks
#include <stdio.h>

int main(void) {
    int x;

    scanf("%d", &x);

    int i = 0;
    while (i < x) {
        int j = 0;
        while (j < x) {
            printf("*");
            j = j + 1;
        }
        i = i + 1;
        printf("\n");
    }

    return 0;
}
```

In other words it should read a numbers and print a square of asterisks of that size.

For example:

```
$ 1521 spim -f square.s
3
***
***
***
$ 1521 spim -f square.s
4
****
****
****
****
$ 1521 spim -f square.s
7
*******
*******
*******
*******
*******
*******
*******
```

Assumptions/Limitations/Clarifications

You can assume the number read is positive (> 0).

When you think your program is working you can `autotest` to run some simple automated tests:

```
$ 1521 autotest square
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs1521 test05_square square.s
```

Sample solution for `square.s`

```
# print a square of asterisks
# x in register $t0
# i in register $t1
# j in register $t2
main:

    li   $v0, 5         #   scanf("%d", &x);
    syscall             #
    move $t0, $v0

    li   $t1, 0         #  i = 0
loop0:
    bge  $t1, $t0, end0 # while (i < x) {

    li   $t2, 0         #   j = 0
loop1:
    bge  $t2, $t0, end1 #   while (j < x) {

    li   $a0, '*'        #     printf("%c", '*');
    li   $v0, 11
    syscall

    addi $t2, $t2, 1    #     j++

    j    loop1          #   }
end1:

    li   $a0, '\n'       #   printf("%c", '\n');
    li   $v0, 11
    syscall

    addi $t1, $t1, 1    #   i++

    j    loop0          # }

end0:
    li   $v0, 0         # return 0
    jr   $ra
```

Submission

When you are finished each exercise make sure you submit your work by running **give**.

You can run **give** multiple times. Only your last submission will be marked.

Don't submit any exercises you haven't attempted.

If you are working at home, you may find it more convenient to upload your work via give's web interface.

Remember you have until **Tue Oct 27 21:00:00 2020** to complete this test.

Automarking will be run by the lecturer several days after the submission deadline for the test, using test cases that you haven't seen: different to the test cases `autotest` runs for you.

(Hint: do your own testing as well as running `autotest`)

Test Marks

After automarking is run by the lecturer you can view it here the resulting mark will also be available via via give's web interface or by running this command on a CSE machine:

```
$ 1521 classrun -sturec
```

The test exercises for each week are worth in total 1 marks.

The best 6 of your 8 test marks for weeks 3-10 will be summed to give you a mark out of 9.