Week 04 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 **Wed Oct 14 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 Wed Oct 14 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 `test04`, changing to this directory, and fetching the provided code by running these commands:

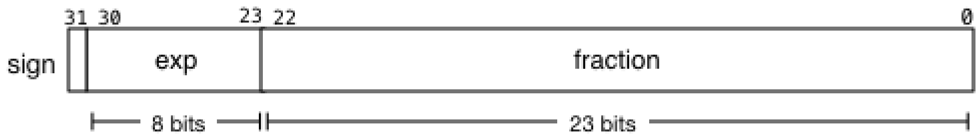
```
$ mkdir test04
$ cd test04
$ 1521 fetch test04
```

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:

Extract the Exponent of A Float

Reminder: Float Representation



Exercise Description

Your task is to add code to this function in **float_exp.c**:

```
// given the 32 bits of a float return the exponent
uint32_t float_exp(uint32_t f) {
    return 42; // REPLACE ME WITH YOUR CODE
}
```

The function `float_exp` is given the bits of a float as type `uint32_t`. Add code so that it returns the exponent.

In other words return the contents of bits 23-30.

This function must be implemented only using bit operations and integer comparisons.

Do not remove the bias (subtract 127) from the exponent.

Once this functions is completed, you should get output like:

```
$ ./float_exp -42
float_exp(-42) returned 0x84
$ ./float_exp 3.14159
float_exp(3.14159012) returned 0x80
$ ./float_exp -inf
float_exp(-inf) returned 0xff
```

Use *make* to build your code:

```
$ make    # or 'make float_exp'
```

Assumptions/Limitations/Clarifications

You may define and call your own functions if you wish.

float_exp should return a value between 0 and 255 inclusive.

You are not permitted to call any functions from the C library, or to perform floating-point operations, or to use variables of type `float` or `double`, or to use multiplication or division.

You are not permitted to change the `main` function you have been given, or to change `float_exp`' prototype (its return type and argument types).

iff stands for "if and only if".

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

```
$ 1521 autotest float_exp
```

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

```
$ give cs1521 test04_float_exp float_exp.c
```

Sample solution for `float_exp.c`

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <assert.h>

#include "float_exp.h"

#define EXPONENT_SHIFT 23
#define EXPONENT_MASK 0xFF

// given the 32 bits of a float return the exponent
uint32_t float_exp(uint32_t f) {
    return (f >> EXPONENT_SHIFT) & EXPONENT_MASK;
}
```

WEEKLY TEST QUESTION:

Flip the Sign of A Float

Reminder: Float Representation

Exercise Description

Your task is to add code to this function in **sign_flip.c**:

```
// given the 32 bits of a float return it with its sign flipped
uint32_t sign_flip(uint32_t f) {
    return 42; // REPLACE ME WITH YOUR CODE
}
```

The function `sign_flip` is given the bits of a float as type `uint32_t`. Add code so that it changes the sign of the number. If it is positive it should be changed to negative. If it is negative it should be changed to positive.

In other words change (flip) bit 31.

This function must be implemented only using bit operations and integer comparisons.

Once this functions is completed, you should get output like:

```
$ ./sign_flip -42
sign_flip(-42) returned 42
$ ./sign_flip 3.14159
sign_flip(3.14159012) returned -3.14159012
$ ./sign_flip -2.718
sign_flip(-2.71799994) returned 2.71799994
$ ./sign_flip -inf
sign_flip(-inf) returned inf
```

Use *make* to build your code:

```
$ make    # or 'make sign_flip'
```

Assumptions/Limitations/Clarifications

You may define and call your own functions if you wish.

You are not permitted to call any functions from the C library, or to perform floating-point operations, or to use variables of type `float` or `double`, or to use multiplication or division.

You are not permitted to change the `main` function you have been given, or to change `sign_flip`' prototype (its return type and argument types).

iff stands for "if and only if".

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

```
$ 1521 autotest sign_flip
```

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

```
$ give cs1521 test04_sign_flip sign_flip.c
```

Sample solution for `sign_flip.c`

```
// Sample solution for COMP1521 lab exercises
//
// Extract the exponent of a float using bit operations only

#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <assert.h>

#include "sign_flip.h"

#define SIGN_MASK (1u << 31)

// given the 32 bits of a float return the exponent
uint32_t sign_flip(uint32_t f) {
    return f ^= SIGN_MASK;
}
```

WEEKLY TEST QUESTION:

Rotate a 16-bit Value N Positions

Download bit_rotate.c, or copy it to your CSE account using the following command:

```
$ cp -n /web/cs1521/20T3/activities/bit_rotate/files.cp/bit_rotate.c .
```

Your task is to add code to this function in **bit_rotate.c**:

```
// return the value bits rotated left n_rotations
uint16_t bit_rotate(int n_rotations, uint16_t bits) {
    return 42; //REPLACE ME WITH YOUR CODE
}
```

The function `bit_rotate` is given an int **n** and a 16-bit value as type **uint16_t**. Add code so that it returns the value with its bits rotated left **n** times.

When rotating a 16-bit value left one position, the value of bit 15 is moved to bit 0, the value of bit 0 to bit 1, the value of 1 to bit 2 ...

For example:

```
$ ./bit_rotate 5 0x0001
bit_rotate(5, 0x0001) returned 0x0020
$ ./bit_rotate 15 0x0001
bit_rotate(15, 0x0001) returned 0x8000
$ ./bit_rotate 18 0x0001
bit_rotate(18, 0x0001) returned 0x0004
$ ./bit_rotate -3 0x0001
bit_rotate(-3, 0x0001) returned 0x2000
$ ./bit_rotate 8 0x1234
bit_rotate(8, 0x1234) returned 0x3412
$ ./bit_rotate 1 0xbeef
bit_rotate(1, 0xbeef) returned 0x7ddf
$ ./bit_rotate 1000 0xdead
bit_rotate(1000, 0xdead) returned 0xadde
$ ./bit_rotate -42 0xfeed
bit_rotate(-42, 0xfeed) returned 0xbb7f
```

Assumptions/Limitations/Clarifications

You may define and call your own functions if you wish.

You are not permitted to call any functions from the C library.

You are not permitted to change the `main` function you have been given, or to change `bit_rotate`'s prototype (its return type and argument types).

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

```
$ 1521 autotest bit_rotate
```

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

```
$ give cs1521 test04_bit_rotate bit_rotate.c
```

Sample solution for `bit_rotate.c`

```c
#include "bit_rotate.h"

uint16_t bit_rotate(int n_rotations, uint16_t bits) {
    uint32_t bits32 = bits;
    n_rotations = n_rotations % 16;
    if (n_rotations < 0) {
        n_rotations += 16;
    }
    bits32 <<= n_rotations;
    return (bits32 & 0xffff) | (bits32 >> 16);
}
```

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 **Wed Oct 14 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.