Final Exam

Exam Conditions

- You can start reading this exam at **Wednesday 19 August 12:50** Sydney time.
- You can start typing at **Wednesday 19 August 13:00** Sydney time.
- You have until **Wednesday 19 August 16:00** Sydney time to complete this exam
- Only submissions before **Wednesday 19 August 16:00** Sydney time will be marked
- You are not permitted to communicate (email, phone, message, talk, ...) with anyone during this exam, except COMP1521 staff via **cs1521.exam@cse.unsw.edu.au**
- You are not permitted to get help from anyone but COMP1521 staff during this exam.
- This is a closed book exam.
- You are not permitted to access papers or books.
- You are not permitted to access files on your computer or other computers, except the files for the exam.
- You are not permitted to access web pages or other internet resources, except the web pages for the exam and the online language cheatsheets & documentation linked below
- **Deliberate violation of exam conditions will be referred to Student Integrity as serious misconduct**

Language Documentation

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

- C quick reference
- MIPS Quick Reference Card
- MIPS Instruction Reference
- SPIM Documentation
- SPIM Alt. Documentation

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

Answer each question in a **SEPARATE** file. Each question specifies the name of the file to use. These are named after the corresponding question number,

Make sure you use **EXACTLY** this file name.

Submit these files using the **give** command as described in each question. **You may submit your answers as many times as you like.** The last submission **ONLY** will be marked.

Ensure that you have submitted your files before the exam deadline.

Running autotests does not automatically submit your code.

Special Considerations

> This exam is covered by the Fit-to-Sit policy. That means that by sitting this exam, you are declaring yourself well enough to do so. You will be unable to apply for special consideration after the exam for circumstances affecting you before it began. If you have questions, or you feel unable to complete the exam, contact **cs1521.exam@cse.unsw.edu.au**

If you experience a technical issue before or during the exam, you should follow the following instructions:

Take screenshots of as many of the following as possible:

- error messages
- screen not loading
- timestamped speed tests
- power outage maps
- messages or information from your internet provider regarding the issues experienced

You should then get in touch with us via **cs1521.exam@cse.unsw.edu.au**

Getting Started

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

```
$ mkdir -m 700 exam_final
$ cd exam_final
$ 1521 fetch exam_final
```

Or you can download the provided code as a zip file or a tar file.

Question 1 (10 MARKS)

You have been given `final_q1.s`, a MIPS assembler program that reads one number and then prints it.

Add code to `final_q1.s` to make it equivalent to this C program:

```c
// print the sum of two integers

#include <stdio.h>

int main(void) {
    int x, y;

    scanf("%d", &x);
    scanf("%d", &y);
    printf("%d\n", x + y);

    return 0;
}
```

In other words, it should read 2 numbers and print their sum.

For example:

```
$ 1521 spim -f final_q1.s
Loaded: /home/cs1521/share/spim/exceptions.s
5
8
13
$ 1521 spim -f final_q1.s
Loaded: /home/cs1521/share/spim/exceptions.s
118
26
144
$ 1521 spim -f final_q1.s
Loaded: /home/cs1521/share/spim/exceptions.s
42
42
84
```

NOTE:

No error checking is required.

Your program can assume its input always contains two integers, and only two integers.

You can assume the value of the expression can be represented as a signed 32 bit value. In other words, you can assume overflow/underflow does not occur.

Your solution must be in MIPS assembler only.

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

```
$ 1521 autotest final_q1
```

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

```
$ give cs1521 final_q1 final_q1.s
```

Question 2 (9 MARKS)

Your task is to add code to this function in **final_q2.c**:

```c
// given a uint32_t value,
// return 1 iff the least significant (bottom) byte
// is equal to the 2nd least significant byte; and
// return 0 otherwise
int final_q2(uint32_t value) {
    // PUT YOUR CODE HERE

    return 42;
}
```

Add code to the function `final_q2` so that, given a `uint32_t` value, it returns **1** iff (if and only if) the least significant (bottom) byte of `value` is equal to the second least significant byte. `final_q2` should return **0** otherwise.

For example, given the hexadecimal value **0x12345678**, `final_q2` should return **0**, because the least significant byte, **0x78**, *is not* equal to the second least significant byte, **0x56**.

Similarly, given the hexadecimal value **0x12345656**, `final_q2` should return **1**, because the least significant byte, **0x56**, *is* equal to the second least significant byte, **0x56**.

You must use bitwise operators to implement `final_q2`.

For example:

```
$ dcc final_q2.c test_final_q2.c -o final_q2
$ ./final_q2 0x00000000
final_q2(0x00000000) returned 1
$ ./final_q2 0x00000001
final_q2(0x00000001) returned 0
$ ./final_q2 0x00000100
final_q2(0x00000100) returned 0
$ ./final_q2 0x00000101
final_q2(0x00000101) returned 1
$ ./final_q2 0x00001212
final_q2(0x00001212) returned 1
$ ./final_q2 0x00001213
final_q2(0x00001213) returned 0
$ ./final_q2 0x12345678
final_q2(0x12345678) returned 0
$ ./final_q2 0x12345656
final_q2(0x12345656) returned 1
$ ./final_q2 0x12345634
final_q2(0x12345634) returned 0
$ ./final_q2 0x12345666
final_q2(0x12345666) returned 0
$ ./final_q2 0x12121212
final_q2(0x12121212) returned 1
$ ./final_q2 0x37861212
final_q2(0x37861212) returned 1
$ ./final_q2 0x12121221
final_q2(0x12121221) returned 0
```

You can also use *make(1)* to build your code:

```
$ make final_q2
```

> **NOTE:**
>
> You are not permitted to call any functions from the C standard library.
>
> You are not permitted to use division (`/`), multiplication (`*`), or modulus (`%`).
>
> You are not permitted to change the `main` function you have been given.
>
> You are not permitted to change `final_q2`'s prototype (its return type and argument types).
>
> No error checking is necessary.
>
> You may define and call your own functions if you wish.

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

```
$ 1521 autotest final_q2
```

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

```
$ give cs1521 final_q2 final_q2.c
```

Question 3 (9 MARKS)

You have been given `final_q3.s`, a MIPS assembler program that reads an integer value and then prints it.

Add code to the file `final_q3.s` so that it prints **1** iff the least significant (bottom) byte of value is equal to the second least significant byte, and prints **0** otherwise.

For example, given the decimal value **305419896**, which is hexadecimal **0x12345678**, `final_q3.s` should print **0**, because the least significant byte, **0x78**, *is not* equal to the second least significant byte, **0x56**.

Similarly, given the decimal value **305419862**, which is hexadecimal **0x12345656**, `final_q3.s` should print **1**, because the least significant byte, **0x56**, *is* equal to the second least significant byte, **0x56**.

For example:

```
$ 1521 spim -f final_q3.s
Loaded: /home/cs1521/share/spim/exceptions.s
0
1
$ 1521 spim -f final_q3.s
Loaded: /home/cs1521/share/spim/exceptions.s
1
0
$ 1521 spim -f final_q3.s
Loaded: /home/cs1521/share/spim/exceptions.s
256
0
$ 1521 spim -f final_q3.s
Loaded: /home/cs1521/share/spim/exceptions.s
257
1
$ 1521 spim -f final_q3.s
Loaded: /home/cs1521/share/spim/exceptions.s
4626
1
$ 1521 spim -f final_q3.s
Loaded: /home/cs1521/share/spim/exceptions.s
4627
0
$ 1521 spim -f final_q3.s
Loaded: /home/cs1521/share/spim/exceptions.s
305419896
0
$ 1521 spim -f final_q3.s
Loaded: /home/cs1521/share/spim/exceptions.s
305419862
1
```

> **NOTE:**
>
> Your solution must be in MIPS assembler only.
>
> Your program can assume its input always contains one integer.
>
> No error checking is necessary.
>
> It is recommended, but not required, that you use bitwise operators.

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

```
$ 1521 autotest final_q3
```

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

```
$ give cs1521 final_q3 final_q3.s
```

Question 4 (9 MARKS)

You have been given `final_q4.s`, a MIPS assembler program that reads 1 number and then prints it.

Add code to `final_q4.s` to make it equivalent to this C program:

```
//  read numbers until their sum is >= 42, print their sum

#include <stdio.h>

int main(void) {
    int sum = 0;
    while (sum < 42) {
        int x;
        scanf("%d", &x);
        sum += x;
    }
    printf("%d\n", sum);
    return 0;
}
```

In other words, it should read numbers until their sum is ≥ 42 and then print their sum.

For example:

```
$ 1521 spim -f final_q4.s
Loaded: /home/cs1521/share/spim/exceptions.s
10
20
25
55
$ 1521 spim -f final_q4.s
Loaded: /home/cs1521/share/spim/exceptions.s
20
22
42
$ 1521 spim -f final_q4.s
Loaded: /home/cs1521/share/spim/exceptions.s
100
100
$ 1521 spim -f final_q4.s
Loaded: /home/cs1521/share/spim/exceptions.s
10
10
10
10
10
50
```

NOTE:

No error checking is required.

Your program can assume its input contains only integers.

Your program can assume these integers sum to ≥ 42.

You can assume the value of the expression can be represented as a signed 32 bit value. In other words, you can assume overflow/underflow does not occur.

Your solution must be in MIPS assembler only.

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

```
$ 1521 autotest final_q4
```

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

```
$ give cs1521 final_q4 final_q4.s
```

Question 5 (9 MARKS)

Write a C program, `final_q5.c`, which takes two names of environment variables as arguments. It should print **1** iff both environment variables are set to the same value. Otherwise, if the environment variables differ in value, or either environment variable is not set, it should print **0**.

The shell command `export` sets an environment variable to a value; the shell command `unset` unsets an environment variable. In the following example, `export` is used to set the environment variables `VAR1`, `VAR2` and `VAR3`, and `unset` is used to ensure environment variables `VAR4` and `VAR5` are unset.

```
$ export VAR1=hello
$ export VAR2=good-bye
$ export VAR3=hello
$ unset VAR4
$ unset VAR5
$ ./final_q5 VAR1 VAR2
0
$ ./final_q5 VAR3 VAR1
1
$ ./final_q5 VAR2 VAR4
0
$ ./final_q5 VAR4 VAR5
0
```

> NOTE:
>
> There is no supplied code for this question.
>
> Your program can assume it is always given 2 arguments.
>
> Your program should always print one line of output. The line of output should contain only 0 or 1.
>
> Your solution must be in C only.
>
> You are not permitted to run external programs. You are not permitted to use system, popen, posix_spawn, fork or exec.

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

```
$ 1521 autotest final_q5
```

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

```
$ give cs1521 final_q5 final_q5.c
```

Question 6 (9 MARKS)

We need to count the number of ASCII bytes in files.

Write a C program, `final_q6.c`, which takes a single filename as its argument, counts the number of bytes in the named file which are valid ASCII, and prints one line of output containing that count.

Assume a byte is valid ASCII iff it contains a value between 0 and 127 inclusive.

- You must match the output format in the example below exactly.

```
$ dcc final_q6.c -o final_q6
$ echo hello world >file1
$ ./final_q6 file1
file1 contains 12 ASCII bytes
$  echo -e 'hello\xBAworld' >file2
$ ./final_q6 file2
file2 contains 11 ASCII bytes
$ echo -n -e '\x80\x81' >file3
$ ./final_q6 file3
file3 contains 0 ASCII bytes
```

> NOTE:
>
> There is no supplied code for this question

There is no supplied code for this question.

No error checking is required.

Your program can assume it is always given the name of a file.

Your solution must be in C only.

You are not permitted to run external programs. You are not permitted to use system, popen, posix_spawn, fork or exec.

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

```
$ 1521 autotest final_q6
```

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

```
$ give cs1521 final_q6 final_q6.c
```

Question 7 (9 MARKS)

You have been given `final_q7.s`, a MIPS assembler program that reads 1 number and then prints it.

Add code to `final_q7.s` to make it equivalent to this C program:

```c
// Read numbers into an array until their sum is >= 42
// then print the numbers in reverse order

#include <stdio.h>

int numbers[1000];

int main(void) {
    int i = 0;
    int sum = 0;
    while (sum < 42) {
        int x;
        scanf("%d", &x);
        numbers[i] = x;
        i++;
        sum += x;
    }

    while (i > 0) {
        i--;
        printf("%d\n", numbers[i]);
    }
}
```

In other words, it should read numbers until their sum is ≥ 42, and then print the numbers read in reverse order.

For example:

```
$ 1521 spim -f final_q7.s
Loaded: /home/cs1521/share/spim/exceptions.s
11
17
3
19
19
3
17
11
$ 1521 spim -f final_q7.s
Loaded: /home/cs1521/share/spim/exceptions.s
10
20
30
30
20
10
$ 1521 spim -f final_q7.s
Loaded: /home/cs1521/share/spim/exceptions.s
42
42
```

No error checking is required.

Your program can assume its input contains only integers.

Your program can assume these integers sum to ≥ 42.

Your program can assume that it will have to read no more than 1000 integers before their sum is ≥ 42.

You can assume the value of the expression can be represented as a signed 32 bit value. In other words, you can assume overflow/underflow does not occur.

Your solution must be in MIPS assembler only.

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

```
$ 1521 autotest final_q7
```

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

```
$ give cs1521 final_q7 final_q7.s
```

Question 8 (9 MARKS)

We need to count the number of UTF-8 characters in a file, and check that the file contains only valid UTF-8.

Write a C program, `final_q8.c`, which takes a single filename as its argument, counts the number of UTF-8 characters in the named file, and prints one line of output containing that count.

Use the same format as the example below.

```
$ dcc final_q8.c -o final_q8
$ echo hello world >file1
$ ./final_q8 file1
file1: 12 UTF-8 characters
$ echo -e -n '\xF0\x90\x8D\x88' >file2
$ ./final_q8 file2
file2: 1 UTF-8 characters
$ echo -e -n '\x24\xC2\xA2\xE0\xA4\xB9' >file3
$ ./final_q8 file3
file3: 3 UTF-8 characters
$ ./final_q8 utf8.html
utf8.html: 7943 UTF-8 characters
```

If `final_q8.c` reads a byte which is not valid UTF-8, it should stop and print an error message in the same format as the example below. Note the error message includes how many valid UTF-8 characters have been previously read.

```
$ echo -e -n '\x24\xC2\xA2\xE0\xA4\x09' >file4
$ ./final_q8 file4
file4: invalid UTF-8 after 2 valid UTF-8 characters
```

If the end of file is reached before the completion of a UTF-8 character, `final_q8.c` should also print an error message; for example:

If the end of file is reached before the completion of a UTF-8 character, `final_q8.c` should also print an error message, for example:

```
$ echo -e -n '\x24\xC2\xA2\xE0\xA4' >file5
$ ./final_q8 file5
file5: invalid UTF-8 after 2 valid UTF-8 characters
```

A reminder of how UTF-8 is encoded:

#bytes	#bits	Byte 1	Byte 2	Byte 3	Byte 4
1	7	`0xxxxxxx`	-	-	-
2	11	`110xxxxx`	`10xxxxxx`	-	-
3	16	`1110xxxx`	`10xxxxxx`	`10xxxxxx`	-
4	21	`11110xxx`	`10xxxxxx`	`10xxxxxx`	`10xxxxxx`

NOTE:

There is no supplied code for this question.

Only the specified error checking is required; no additional error checking is required.

Your program can assume it is always given a single argument, the name of a file.

Your solution must be in C only.

You are not permitted to run external programs. You are not permitted to use system, popen, posix_spawn, fork or exec.

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

```
$ 1521 autotest final_q8
```

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

```
$ give cs1521 final_q8 final_q8.c
```

Question 9 (9 MARKS)

You have been given `final_q9.s`, a MIPS assembler program that reads a line of input and then prints 42.

Add code to `final_q9.s` to make it equivalent to this C program:

```c
#include <stdio.h>

char *s;

int expression(void);
int term(void);
int number(void);

int main(int argc, char *argv[]) {
    char line[10000];
    fgets(line, 10000, stdin);
    s = line;
    printf("%d\n", expression());
    return 0;
}

int expression(void) {
    int left = term();
    if (*s != '+') {
        return left;
    }
    s++;
    int right = expression();
    return left + right;
}


int term(void) {
    int left = number();
    if (*s != '*') {
        return left;
    }
    s++;
    int right = term();
    return left * right;
}

int number(void) {
    int n = 0;
    while (*s >= '0' && *s <= '9') {
        n = 10 * n + *s - '0';
        s++;
    }
    return n;
}
```

The above C program reads a line of input containing an arithmetic expression and prints its value.

The arithmetic expression contains only positive integers, and multiply (`'*'`)and add (`'+'`) operators.

For example:

```
$ 1521 spim -f final_q9.s
Loaded: /home/cs1521/share/spim/exceptions.s
6*7
42
$ 1521 spim -f final_q9.s
Loaded: /home/cs1521/share/spim/exceptions.s
1+2*3*4+5
30
$ 1521 spim -f final_q9.s
Loaded: /home/cs1521/share/spim/exceptions.s
100+2*20+978
1118
$ 1521 spim -f final_q9.s
Loaded: /home/cs1521/share/spim/exceptions.s
1000+777+66+55*444+9
26272
```

> **NOTE:**
>
> No error checking is required.

Your program can assume it is given *exactly* one line of input.

Your program can assume this line contains only these 12 characters: **0123456789+***

You can assume the line contains less than 10,000 characters.

Your program can assume that these characters form a valid arithmetic expression.

You can assume the value of the expression can be represented as a signed 32 bit value. In other words you can assume overflow does not occur.

Your solution must be in MIPS assembler only.

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

```
$ 1521 autotest final_q9
```

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

```
$ give cs1521 final_q9 final_q9.s
```

Question 10 (9 MARKS)

We need a program which will save an entire directory tree as a single file. The directory tree may consist of many directories and files.

Write a C program, `final_q10`, which is given either 1 or 2 arguments.

If `final_q10` is given 2 arguments, the first argument will be the pathname of a file it should create and the second argument will be the pathname of a directory. `final_q10` should then save the entire contents of the specified directory tree in the specified file.

If `final_q10` is given 1 argument, that argument will be the pathname of a file in which a directory tree has been saved. `final_q10` should re-create all the directories and files in the directory tree.

For example, these commands create a directory tree named **a**:

```
$ mkdir -p a/b/c
$ echo hello andrew >a/file1
$ echo bye andrew >a/b/file2
$ echo 1 >a/b/c/one
$ echo 2 >a/b/c/two
$ ls -lR a
a:
total 8
drwxr-xr-x 3 z1234567 z1234567 4096 Aug 19 20:38 b
-rw-r--r-- 1 z1234567 z1234567   13 Aug 19 20:38 file1

a/b:
total 8
drwxr-xr-x 2 z1234567 z1234567 4096 Aug 19 20:38 c
-rw-r--r-- 1 z1234567 z1234567   11 Aug 19 20:38 file2

a/b/c:
total 8
-rw-r--r-- 1 z1234567 z1234567 2 Aug 19 20:38 one
-rw-r--r-- 1 z1234567 z1234567 2 Aug 19 20:38 two
$ cat a/file1
hello andrew
$ cat a/b/file2
bye andrew
```

In this example, `final_q10` saves the contents of the directory tree **a** into a file named **data**:

```
$ dcc final_q10.c -o final_q10
$ ./final_q10 data a
$ ls -l data
-rw-r--r-- 1 z1234567 z1234567 4567 Aug 19 20:38 data
```

This example shows `final_q10` restoring the contents of the directory tree **a** after it has been removed:

```
$ rm -rf a
$ ls -lR a
ls: cannot access 'a': No such file or directory
$ cat a/file1
cat: a/file1: No such file or directory
$ ./final_q10 data
$ ls -lR a
a:
total 8
drwxr-xr-x 3 z1234567 z1234567 4096 Aug 19 20:38 b
-rw-r--r-- 1 z1234567 z1234567   13 Aug 19 20:38 file1

a/b:
total 8
drwxr-xr-x 2 z1234567 z1234567 4096 Aug 19 20:38 c
-rw-r--r-- 1 z1234567 z1234567   11 Aug 19 20:38 file2

a/b/c:
total 8
-rw-r--r-- 1 z1234567 z1234567 2 Aug 19 20:38 one
-rw-r--r-- 1 z1234567 z1234567 2 Aug 19 20:38 two
$ cat a/file1
hello andrew
$ cat a/b/file2
bye andrew
```

This example shows `final_q10` restoring the contents of the directory tree **a** in a different directory:

```
$ mkdir new_directory
$ cd new_directory
$ ../final_q10 ../data
$ cat a/file1
hello andrew
$ cat a/b/file2
bye andrew
```

WARNING:

Autotest will only be of limited assistance in debugging your program. Do not expect autotest messages to be easy to understand for this problem. You will need to debug your program yourself.

DANGER:

It is easily possible to destroy many files with *rm(1)*. Do not test this on your working files for this exam, unless you have a backup. Do not assume your program will make a good enough backup.

NOTE:

Your solution must be in C only.

You are not permitted to run external programs. You are not permitted to use system, popen, posix_spawn, fork or exec.

You are not permitted to use libraries other than the default C libraries. In other words your solution can not require use of dcc's -l flag. If your solution compiles without dcc's -l flag, you are using only the default C libraries. All functions discussed in lectures are part of the default C libraries.

No error checking is necessary.

You can assume the directory tree to be saved contains only directories and regular files. You can assume it does not contain links or other special files. You can assume it does not contain sparse files.

You can not assume anything about the size or contents of files in the directory tree. The files may contain any byte. The files may be any size.

Your program does not have to save or restore permissions, modification times, or other file metadata.

You can assume the directory tree to be saved contains at most 10000 directories and regular files.

You can assume the directory tree to be saved is at most 1000 levels deep.

You can assume files and directories do not already exist when restoring a directory tree.

> You can use any format you choose to save the directory tree in the file.

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

```
$ 1521 autotest final_q10
```

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

```
$ give cs1521 final_q10 final_q10.c
```

Question 11 (9 MARKS)

You have been given `final_q11.s` a MIPS assembler program that reads a line of input and then prints 42.

You have been given `final_q11.s`, a MIPS assembler program that reads a line of input and then prints 42.

Add code to `final_q11.s` to evaluate the line as an arithmetic expression. and print its value.

The arithmetic expression will contain only positive integers and multiply (`'*'`) and add (`'+'`) operators.

The integers may be arbitrarily large; except they must fit on a line of less than 10,000 characters.

There are no marks for approaches which handle only smaller integers; for example, those small enough to be represented in only 32 or 64 bits.

There are no marks for approaches which use floating point arithmetic or other methods to produce approximate results.

For example:

```
$ 1521 spim -f final_q11.s
Loaded: /home/cs1521/share/spim/exceptions.s
982451653*961748941*472882027*452390477
202133905243937086663174722891174767
$ 1521 spim -f final_q11.s
Loaded: /home/cs1521/share/spim/exceptions.s
2*1000000000000000000000000000000000000000000000000000000000000000000000000000+3*4
200000000000000000000000000000000000000000000000000000000000000000000000000012
```

> **NOTE:**
>
> No error checking is required.
>
> Your program can assume it is given *exactly* one line of input.
>
> Your program can assume this line is made up of only these 12 characters: `0123456789+*`
>
> You can assume the line contains less than 10,000 characters.
>
> Your program can assume that these characters form a valid arithmetic expression.
>
> Your solution must be in MIPS assembler only.
>
> You are not permitted to use floating point operations or registers.

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

```
$ 1521 autotest final_q11
```

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

```
$ give cs1521 final_q11 final_q11.s
```

Submission

When you are finished each questions 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 questions you haven't attempted.

You can check if you have made a submisison like this:

```
$ 1521 classrun -check final_q1
```

Remember you have until **Wednesday 19 August 16:00** Sydney time to complete this exam (not including any extra time provided by ELS conditions).

Do your own testing as well as running `autotest`

COMP1521 20T2: Computer Systems Fundamentals is brought to you by
the School of Computer Science and Engineering
at the University of New South Wales, Sydney.
For all enquiries, please email the class account at cs1521@cse.unsw.edu.au
CRICOS Provider 00098G