University of St Andrews

2021/2 SEMESTER 2
8 HOUR TAKE HOME EXAM
SCHOOL OF COMPUTER SCIENCE

MODULE CODE: **CS2002**

MODULE TITLE: Computer Systems

TIME TO HAND IN: 8 hours

INSTRUCTIONS:

(a) Answer **all** questions.

(b) Each question indicates the number of marks it carries. The paper carries a total of 60 marks.

This assessment consists of exam-style questions and you should answer as you would in an exam. You cannot copy or paraphrase text or material from other sources and present this as your own work. Your exam answers should be entirely your own work without unacknowledged input from others. If you are in any doubt, you should clearly acknowledge the origin of any material, text passages or ideas presented (e.g. through references). You must not co-operate with any other person when completing the exam, which must be entirely your own work. You must not share any information about the exam with another person (e.g. another student) or act on any such information you may receive. Any attempt to do so will be dealt with under the University's Policy for Good Academic Practice and may result in severe sanctions. You must submit your completed assessment on MMS within 8 hours of you downloading the exam. Assuming you have revised the module contents beforehand, answering the questions should take no more than three hours.

You will not be penalised for minor syntax errors in programming questions.

1. Architecture

   (a) A CPU with a 5 stage pipeline might allow a 5 times shorter clock cycle, but instructions would take 5 clock cycles to complete, so any individual instruction may not complete any faster. What then is the key advantage that CPUs obtain from pipelining? Given the advantages suggest two reasons for not increasing the number of stages to a much higher number than 5.

   [3 marks]

   (b) Suppose we are using direct mapped cache with a CPU that addresses 8-bit bytes using a 32 bit address space, with a cache containing 4096 cache lines of 512 bytes each. How many bits will be used for the offset, index, and tag? How many bytes of memory are necessary to store the cache, including the supporting information needed for its correct operation? Give the binary representations of the index, tag, and offset of the memory address `00D36E64` (written in hexadecimal). [8 marks]

   (c) In 1993 the first Intel Pentium chip – a very succesful CPU with a single core – had about 3.1 million transistors on it. In 2022 a modern chip such as the Apple M1 Pro has about 34 billion transistors, more than 10,000 times as many, but has only 10 cores. Why don't modern hardware designers simply design a chip to be 10,000 copies of a Pentium-like core? Suggest **TWO** reasons why this is not done and that the current approach is followed, explaining your answers. [4 marks]

   [Total marks 15]

2. Logic

(a) Suppose we want to build a 'universal logic circuit' with three boolean data input bits. How many control input bits would be necessary for this circuit to be able to perform as any three variable boolean function? Could we use a multiplexer for this circuit? If so, explain how we would do so, or if not explain why we could not. [3 marks]

(b) Consider the following truth table as the intended behaviour of a circuit with three inputs A, B, C, and one output X.

A	B	C	X
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Draw a circuit which correctly implements the above truth table using gates selected from AND, OR, NOT, XOR, NAND, and NOR gates. Give evidence to show that your circuit accurately matches the truth table. Full credit will be given for any correct circuit involving two gates or fewer, with partial credit for larger correct circuits. [8 marks]

(c) Consider the n-bit ripple adder described in lectures. Give, as a function of n, the number of gates needed to construct an n-bit ripple adder and the maximum number of gate delays between the inputs and the outputs. Explain your answers. What are the tradeoffs involved in designing an n-bit adder using a different technique which would have fewer gate delays? [4 marks]

[Total marks 15]

3. Dynamic Memory Allocation and Structs

Consider the C module interface below which is part of a fixed-size, dynamically allocated array based `Collection` module that holds pointers to arbitrary elements.

```
typedef struct Collection {
    // Your struct definition here
} Collection;

// Returns a pointer to a new Collection of given max_size.
Collection* new_Collection(int max_size);

// Returns true on successfully adding the element to this Collection.
bool Collection_add(Collection* this, void* element);

// Returns the number of elements in this collection.
int Collection_size(Collection* this);

// Frees the memory used by this collection.
void Collection_free(Collection* this);
```

(a) Provide a suitable definition of the `Collection` struct. [2 marks]

(b) Implement the `new_Collection` function, taking care to deal with `NULL`s where appropriate. [5 marks]

(c) Provide implementations of the `Collection_add`, `Collection_size` and `Collection_free` functions (state any assumptions you make). [4 marks]

(d) Now consider implementing a print function for your collection which is supposed to print out relevant detail of all elements that are in a collection depending on the type of element. Explain how this might be achieved in C using suitable structs. Outline the implementation of your `Collection_print` function and how you would alter the `Point` struct shown below so that your `Collection_print` function would be able to print out the `x` and `y` coordinates for `Point` types.

```
// Prints out relevant information of all elements in this collection.
void Collection_print(Collection* this);

// Struct representing a point in 2-D space
typedef struct Point {
    int x, y;
} Point;
```

[4 marks]

[Total marks 15]

4. Concurrent Computation

Consider the `computeSum` function shown below, which computes and returns the sum of the integers in the `array` of given `size`.

```
int computeSum(int array[], int size) {
    int sum = 0;
    for (int i = 0; i < size; i++) {
        sum += array[i];
    }
    return sum;
}
```

In order to exploit parallelism on multi-core processors, you are to develop parallel versions of the C code in which the summation is split into two tasks (separately summing up array elements of one half of the array and the other half of the array) and these tasks are executed concurrently. You may assume the existence of a suitable `computePartialSum` function as shown below.

```
int computePartialSum(int array[], int low, int high) {
    int sum = 0;
    for (int i = low; i < high; i++) {
        sum += array[i];
    }
    return sum;
}
```

(a) Provide an implementation as indicated above in which each half of the summation is executed in a separate thread and the total sum is afterwards returned from the `computeSum` function. [8 marks]

(b) Now provide an implementation in which each half of the summation is executed in a separate process and the total sum is afterwards returned from the `computeSum` function. [7 marks]

[Total marks 15]

***** END OF PAPER *****