The Australian National University
Mid Semester Examination – April 2021

Comp2300 & Comp6300
Computer Organisation & Program Execution

Study period:	15 minutes
Time allowed:	2 hours (after study period)
Total marks:	50
Permitted materials:	None

Questions are **not** equally weighted – sizes of answer boxes do **not** necessarily relate to the number of marks given for this question.

All your answers must be written in the boxes provided in this exam form. You can use scrap paper for working, but only those answers written in this form will be marked. Do not upload your exam anywhere but the prescribed exam submission system. There is additional space at the end of the booklet in case the answer boxes provided are insufficient. Label any answer you write at the end of the exam form with the number of the question it refers to and note at the question itself, that you provided addition material at the end.

Greater marks will be awarded for answers that are simple, short and concrete than for answers of a sketchy and rambling nature. Marks will be lost for giving information that is irrelevant to a question.

Student number:

The following are for use by the examiners

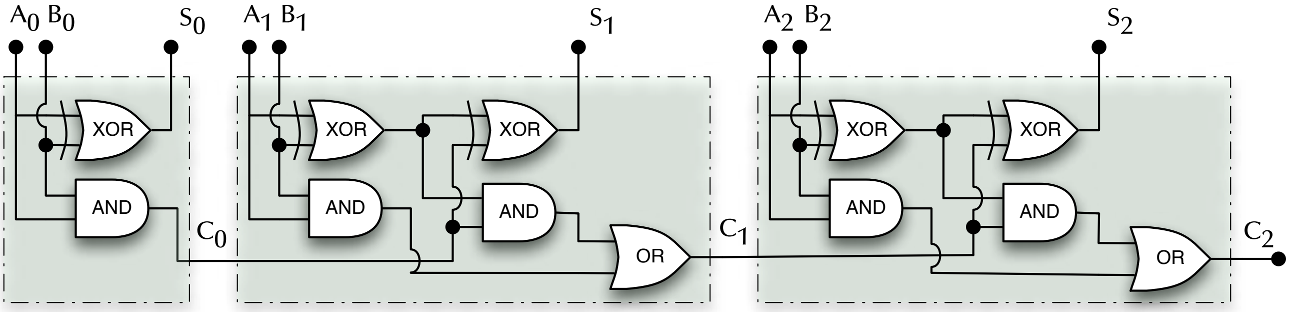
Q1 mark	*Q2 mark*	*Q3 mark*

Total mark

1. [10 marks] Digital Logic / Basic CPU architecture

(a) [5 marks] The following circuit diagram show a correctly working, 3-bit ripple-carry adder.

(i) [2 marks] Use this adder to calculate S_0 to S_2 as well as C_0 to C_2 given the values for A and B in the table below.



Index	A	B	S	C
0	$A_0 = 0$	$B_0 = 0$	$S_0 =$	$C_0 =$
1	$A_1 = 1$	$B_1 = 1$	$S_1 =$	$C_1 =$
2	$A_2 = 0$	$B_2 = 1$	$S_2 =$	$C_2 =$

(ii) [3 mark] What arithmetic operation let to the above results S_0 to S_2 and what do those C_0 to C_2 values tell you? If there are multiple possible options then list them all.

(b) [5 marks] The function $q(a,b,c)$ is defined by the following truth-table. Use minterms *or* maxterms to determine and simplify the output q to the shortest possible form (the smallest number of boolean operators). You can also use algebraic transformations to simplify q. Use text notations like `(x and y)` or `not (x)` if you don't find standard logic operators on your keyboard.

a	b	c	Output q	min- or max terms	Simplified min- or maxterms
F	F	F	F		
F	F	T	T		
F	T	F	T		
F	T	T	F		
T	F	F	T		
T	F	T	F		
T	T	F	F		
T	T	T	T		

$q =$

2. [10 marks] Machine instructions

(a) [5 marks] Compile one of the following code snippets (provided in both C-style and Algol-style syntactical forms for your convenience) into ARM assembly code and keep the value of `Power` in r0.

C-style:

```
unsigned int Power = 1;

int i;
for (i = 1; i < 11; i++) {
  Power = Power * 2;
}
```

Algol-style:

```
declare
   Power : Natural := 1;
begin
   for i in 1 .. 10 loop
      Power := Power * 2;
   end loop;
end;
```

(b) [4 marks] Consider the following ARM code:

```
    ldr   r0, =array
    ldrb  r1, [r0], #1    @ Loads a single byte from memory
loop:
    ldrb  r2, [r0], #1    @ Loads a single byte from memory
    cmp   r1, r2
    beq   terminate       @ eq stands for “equal”
    mov   r1, r2
    b     loop
terminate:
    b     terminate

array:
    .ascii    “Helloooo”
```

(i) [1 mark] How many machine instructions will have been executed from the beginning of this code, until it reaches ***terminate***? If it does not reach it then say “infinite”.

(ii) [1 marks] If the label ***terminate*** is or would be reached: what is the meaning of the value of r0 at that time?

(iii) [3 marks] Rewrite the above code using (index) register-offset addressing (instead of post-indexed addressing). (no need to re-write or copy anything after ***terminate***.)

3. [30 marks] Functions

(a) [15 marks] Read the following functions carefully:

Imperative syntax:

```
function a (x : Natural) return Natural is

  function b (y : Natural) return Natural is

    function c (z : Natural) return Natural is

      (if z > 0 then b (z - 1) else x + y);

  begin
    return (if y > 0 then c (y - 1) else x);
  end b;

begin
  return b (x);
end a;
```

Functional syntax:

```
a ::  Natural ->  Natural
a x = b x

  where

  b :: Natural -> Natural
  b y = if y > 0 then c (y - 1) else x

    where

    c :: Natural -> Natural
    c z = if z > 0 then b (z - 1) else x + y
```

The functions are identical in both syntax forms. In both cases function c is defined inside of function b, which itself is defined inside of function a.

Continue to the questions on the next page.

(i) [2 marks] If function a is called with a parameter value x of 5 (i.e. you write `a (5)`), what will be the value of x and y as seen from inside the function c, when c evaluates the return value `x + y`?

(ii) [3 marks] In the examples above, both compilers will produce an error message, if function b would attempt to access the parameter z of function c. Is this a chosen restriction by those programming languages (e.g. to avoid messy programming), or is it actually impossible to provide such an access? Explain as precisely as you can.

(iii) [5 marks] What code will need to be produced by your compilers to provide the correct access to x and y as seen from inside the function c? To answer the question in full, you do not need to write out the actual assembler code. It is sufficient to describe the essential mechanisms which are required – but do this as precisely as you can.

(iv) [5 marks] List everything what you could potentially find inside a stack frame for function b. Explain the meaning and/or usage of each item precisely.

(b) [15 marks] The local variables inside a function are given below. Two different syntactical styles are provided for you, expressing in both cases that you have a product type (also known as “record” or “struct”) with 3 natural number fields for colour channels, and a single natural number for brightness.

C-style:

```
typedef struct Colours
{
  unsigned int Red, Green, Blue;
} Colours;

// Local variables:

Colours Color;
unsigned int Brightness;
```

Algol-style:

```
type Colours is record
   Red, Green, Blue : Natural;
end record;

-- Local Variables:

Colour : Colours;
Brightness : Natural;
```

(i) [2 marks] Write ARM assembly code which allocates memory space for those variables and explain your answer.

(ii) [3 marks] When does this allocation code need to be executed and how and when are those local variables de-allocated again? Give precise answers.

(iii) [5 marks] Write ARM assembly code which initialises those local variables according to the statements below. Assume that `FP - 12` is the address of the first word of those local variables.

C-style:

```
Color.Red   = 1;
Color.Green = 2;
Color.Blue  = 3;

Brightness = Color.Red
           + Color.Green
           + Color.Blue;
```

Algol-style:

```
Colour := (Red   => 1,
           Green => 2,
           Blue  => 3);

Brightness := Colour.Red
            + Colour.Green
            + Colour.Blue;
```

(iv) [5 marks] Write a complete ARM assembly function which takes a colour (as defined by `Colours` above) as a parameter and returns the brightness (as defined above by `Brightness`) as the sum of the three colour components.

continuation of answer to question □ *part* □

continuation of answer to question □ *part* □