The University of Nottingham

SCHOOL OF COMPUTER SCIENCE

A SAMPLE EXAM PAPER

COMPUTER SYSTEMS ARCHITECTURE

(Course G51CSA)

Time allowed ONE hour

Candidates may complete the front cover of their answer book and sign their desk card but must NOT write anything else until the start of the examination period is announced

Answer any *TWO* questions

Marks available for sections of questions are shown in brackets in the right-hand margin

No calculators are permitted in this examination

Dictionaries are not allowed with one exception. Those whose first language is not English may use a standard translation dictionary to translate between that language and English provided that neither language is the subject of this examination. Subject specific translation dictionaries are not permitted.

No electronic devices capable of storing and retrieving text, including electronic dictionaries, may be used.

DO NOT turn examination paper over until instructed to do so

ADDITIONAL MATERIAL PROVIDED
ARM Assembler Reference (2 pages)

*The questions on this paper that relate to the ARM chip, refer to the 32-bit ARM architecture running in the default 'little endian' mode, as supported by the **aasm** assembler and the KoMoDo environment.*

1 Complete the following table i.e. calculate the 12 values labelled *a–l*

(*a*)

Decimal	Binary	Octal	Hexa-decimal
123	*a*	*b*	*c*
d	1011	*e*	*f*
g	*h*	123	*i*
j	*k*	*l*	123

(12)

(*b*) Calculate the representations of the following decimal numbers in 8-bit two's complement:

$m = 44$

$n = -56$

$o = 77$

(6)

(*c*) Calculate in detail, using 8-bit two's complement, the sums $m + n$, $n - o$, $m + o$. In which case does an overflow occur? (7)

2

(*a*) Explain the difference between an AND gate and an OR gate. (5)

A games console is being designed using a microprocessor with a 16-bit address bus (`A0−A15`). The 64K of memory is to be split and allocated to RAM, ROM and I/O hardware as follows:

Address Range (hex)	Contains	Select Signal
`0X0000−0x002C`	VIDEO I/O	`VIDCS`
`0x0100−0x01FF`	RAM	`RAMCS`
`0x2010−0x2012`	I/O	`IOCS`
`0xF000−0xFFFF`	Cartridge ROM	`ROMCS`

The rest of the address space is unused.

Note: As with many computer systems, it its only necessary to decode the address to sufficiently identify each of the four stages above. It is acceptable for some parts to be decodable by more than one address ***provided*** *these extra addresses do not overlap any of the other specified address ranges.*

(*b*) Using a combination of *AND*, *OR* and *NOT* gates give logic equations for the signals `VIDCS`, `RAMCS`, `IOCS`, and `ROMCS` in the above table such that the signal is true (logic `1`) when the address bus (`A0−A15`) contains an address in the specified range. (12)

(*c*) Give the logic equation for a signal `INVALID` which is true when the address bus contains an address that does not fall into one of the sections listed above. (8)

3 (*a*) Consider the following fragment of an ARM program

```
      ;  input integers are in R1, R2
         MOV R0, #0
         B el
l1       ADD R0, R0, R1
         SUB R2, R2, #1
el       CMP R2, #0
         BNE l1
out   ;  result is in R0
```

Simulate the program for the values of R1, and R2 shown in the table below. Write down, in each case, what the value will be in register R0 when execution reaches the label out, i.e. supply the correct values for *a*, *b* and *c* in the table below. All numbers are in decimal.

R1	R2	R0
1	3	a
5	20	b
25	3	c

Describe in one sentence what the program calculates (8)

(*b*) The following C program fragment calculates the factorial of a positive integer, *x*.

```
y = 1;
for(i = 1; i <= x; i++)
       y = y * i;
```

Translate this program first to a C program using a while loop, then to a C program using goto and finally to a program in ARM assembler which prints the integer answer to the Komodo 'Features' window using a SWI 4 system call. You should load a value for x from location defined in memory using a DEFW directive at the top of the program. (12)

(*c*) Implement a program in ARM assembler which takes a positive integer *x* calculates the result of the following formula for the given *x*

```
y = x * (x + 1) / 2
```

Implement this new solution in ARM assembler code. The chosen value of x should be set up using DEFW, and kept small enough that any multiply operations do not cause overflow. Output your answer using SWI 4. (5)