Cambridge International **A Level**

Cambridge International Examinations
Cambridge International Advanced Level

CANDIDATE NAME	
CENTRE NUMBER	
CANDIDATE NUMBER	

2852209988

COMPUTING **9691/31**

Paper 3 **October/November 2014**

2 hours

Candidates answer on the Question Paper.

No additional materials are required.

No calculators allowed.

READ THESE INSTRUCTIONS FIRST

Write your Centre number, candidate number and name on all the work you hand in.
Write in dark blue or black pen.
You may use a soft pencil for any diagrams, graphs or rough working.
Do not use staples, paper clips, glue or correction fluid.
DO **NOT** WRITE IN ANY BARCODES.

Answer **all** questions.
No marks will be awarded for using brand names for software packages or hardware.

At the end of the examination, fasten all your work securely together.
The number of marks is given in brackets [ ] at the end of each question or part question.

This document consists of **16** printed pages.

CAMBRIDGE International Examinations

1 **(a)** Convert the following infix form expressions into reverse Polish notation.

(i) `(a + b) / 6`

.....[1]

(ii) `3 * (x * y + 3)`

.....[2]

(b) Convert the following reverse Polish notation expressions into infix form.

(i) `3 x y + z + *`

.....[1]

(ii) `7 y ^ 6 + 2 /`

Note: the caret (^) symbol represents “to the power of”.

.....[2]

(c) An expression in reverse Polish notation can be evaluated on a computer system using a stack.

(i) Describe the operation of a stack.

.....

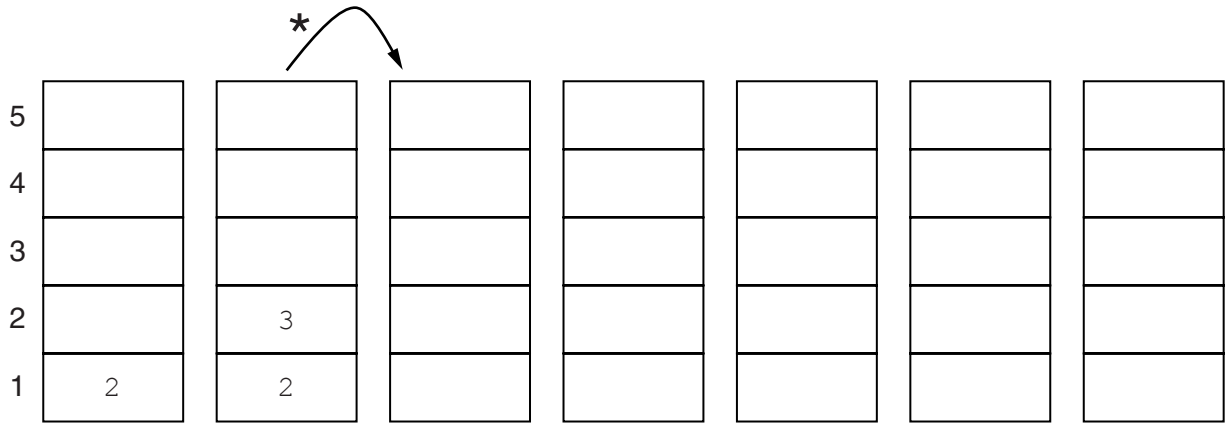
.....[1]

(ii) The expression in reverse Polish notation

`2 3 * 8 + 7 /`

is to be evaluated using a stack. The first available location on the stack is 1.

The diagram will show the changing contents of the stack as this expression is evaluated. Complete the diagram.



[4]

2 **(a)** Modern operating systems use a memory management technique called paging.

Explain how paging works by using the terms:

- Page
- Page frame
- Page table

.....

.....

.....

.....

.....

.....[3]

(b) For a computer system using multi-programming, the low-level scheduler decides what process will get next use of the processor.

One algorithm could be a “round-robin”; that is, every process gets use of the processor in sequence.

State **two** other algorithms which could be used by the low-level scheduler.

1 .....

.....

2 .....

.....[2]

(c) For a "round-robin" algorithm, five processes are currently loaded and get the use of the processor in the sequence:

PROG16 – PROGWP – PROGDB – PROG11 – PROG31, then return to PROG16

Process PROGDB has just completed its time-slice.
Put the sequence of events below in the correct order.

Note: two of the statements will not be used.

A	Process PROG11
B	Interrupt received from the low-level scheduler
C	Copy to the CPU registers the contents of the file directory for PROG11
D	Save the PC and all other registers contents for PROGDB to its Process Control Block (PCB)
E	Copy to the CPU registers the contents of the stack
F	Copy to the CPU registers the contents of the Process Control Block for PROG11

List the sequence of events using the letters.

.....

.....

.....

.....

..... [4]

3 A database is to be set up to store data about paintings sold to customers by a gallery.

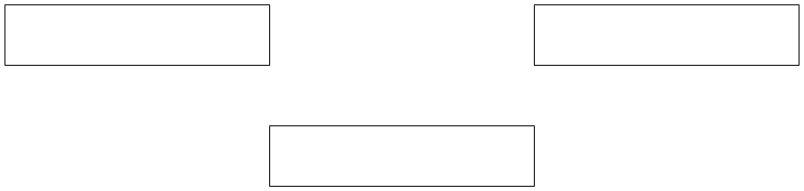
Several attempts are made at the database design.

(a) Consider Design 1:

```
Customer(CustomerID, CustAddress, DateRegistered)
Painting(PaintingID, Description, PaintingDate, Artist, Price)
Sales(SalesID, CustomerID, PaintingID, PurchaseDate)
```

(i) Circle above the two foreign keys in this database design. [2]

(ii) These two foreign keys form two relationships.
Complete the entity-relationship (E-R) diagram to show them.



[2]

(iii) It is suggested that, as the number of sales made is relatively small, a `SalesID` is not required. The `Sales` table could be re-designed as:

```
Sales(CustomerID, PurchaseDate, PaintingID)
```

This design is to be implemented.
How will this restrict the gallery's sales?

.....

.....[1]

(b) More data is to be stored about the artist and the customer.

Consider Design 2:

```
Customer(CustomerID, CustomerName, CustAddress, DateRegistered)
Painting(PaintingID, Description, PaintingDate, ArtistName,
       ArtistDateBorn, ArtistDateDied, ArtistNationality, Price)
Sales(CustomerID, PurchaseDate, CustomerName, PaintingID)
```

(i) Name the table which is not in Second Normal Form (2NF) and explain why.

Table .....

Explanation .....

.....

.....

Re-design this table.

.....[3]

(ii) Name the table which is not in Third Normal Form (3NF) and explain why.

Table .....

Explanation .....

.....

.....

Re-design this table and add a new table. Both these tables must be fully normalised.

.....

.....

.....

.....[5]

(c) The final design for the customer table is:

Customer(CustomerID, CustomerName, CustAddress, TelNo, DateRegistered)

Customer 065 has contacted the gallery to tell them his new telephone number is 0123 456789.

Write the Data Manipulation Language (DML) command to update their record.

.....

.....

.....

.....[3]

4 The table below gives a subset of the assembly language instructions for a computer with a single general-purpose register, the Accumulator (ACC), and an index register (IX).

Instruction			Explanation
Opcode (mnemonic)	**Operand**	**Opcode (binary)**	
LDD	<address>	0000 0100	Direct addressing. Load the contents of the given address to ACC
LDV	<number>	0000 0101	Load the given number to ACC
STO	<address>	0001 0000	Store the contents of ACC at the given address
LDI	<address>	0000 0110	Indirect addressing. At the given address is the address to be used. Load the contents of this second address to ACC
LDX	<address>	0000 0111	Indexed addressing. Form the address as <address> + the contents of IX. Copy the contents of this address to ACC
INC	<register>	0000 0011	Add 1 to the contents of the register (ACC or IX)
OUTCH		1000 0001	Output to the monitor the character corresponding to the ASCII character code in ACC
IN		1001 0000	Input a denary number from the keyboard and store in ACC
JMP	<address>	1100 1000	Unconditional jump to the given address
END		1111 1111	End the program and return to the operating system

The diagram shows a program loaded in main memory starting at location 100.

Locations 200 onwards contain data which are used by the program.

(a) **(i)** The instruction at address 102 is fetched.

ACC

IX

Show the contents of the registers after execution.
Write on the diagram to explain. [2]

100	LDI 150
101	OUTCH
102	LDD 203
103	INC ACC
104	STO 150
105	JP 100
106	END
107	
	≈
150	200
	≈
200	65
201	76
202	65
203	77
204	32
205	32

(ii) The instruction at address 100 is fetched.

ACC

IX

Show the contents of the registers after execution.
Write on the diagram to explain. [3]

(b) The given table of instructions shows the binary number used for each instruction's opcode.

All instructions in machine code are stored as a 16-bit pattern, with the opcode as the first 8 bits and the operand as the second 8 bits.

(i) What is the maximum number of different instructions this processor could have?

.....[1]

(ii) Consider the instruction:

0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1

Describe what this instruction does.

.....

.....[2]

(iii) Programmers prefer to write machine code instructions in hexadecimal.

Explain why.

.....

.....[1]

(iv) What is the hexadecimal number for the instruction shown in **part (b)(ii)**?

..... [1]

Show the machine code for the following instructions:

(v) `LDI 150`

[2]

(vi) `LDV 15`

[2]

(vii) A programmer makes the statement:

"For this instruction set, some of the instructions do not require an operand"

Circle if this statement is true or false and explain with reference to the instructions given.

True / False

.....

.....

.....

.....[2]

(c) Use the ASCII code table to trace the **first four iterations** of the given program.

ASCII code table (part)					
Character	**Decimal**	**Character**	**Decimal**	**Character**	**Decimal**
<Space>	32	I	73	R	82
A	65	J	74	S	83
B	66	K	75	T	84
C	67	L	76	U	85
D	68	M	77	V	86
E	69	N	78	W	87
F	70	O	79	X	88
G	71	P	80	Y	89
H	72	Q	81	Z	90

ACC	Location 150	OUTPUT

```
100  LDI   150
101  OUTCH
102  LDD   150
103  INC   ACC
104  STO   150
105  JP    100
106  END
107
     ~~
150      200
     ~~
200      65
201      76
202      65
203      77
204      32
205      32
```

[5]

5 Most modern computers are designed using Von Neumann architecture.

(a) Describe what is meant by Von Neumann architecture.

.....

.....

.....

.....[2]

(b) The sequence of operations below shows the fetch stage of the fetch-execute cycle in register transfer notation.

```
1. MAR ← [PC]
2. PC ← [PC] + 1
3. MDR ← [[MAR]]
4. CIR ← [MDR]
```

Note: `[register]` denotes the contents of the specified register.

Explain what is happening at the fetch stage.

1 .....

.....

.....

2 .....

.....

.....

3 .....

.....

.....

4 .....

.....

.....[4]

(c) The address bus and data bus are used during the fetch-execute cycle.

(i) Name another bus used in a typical microprocessor.

..... [1]

(ii) Name **one** signal carried by this bus.

.....

.....[1]

(d) Consider **two** assembly language instructions which were given in **Question 4**.

Instruction		Explanation
Opcode (mnemonic)	**Operand**	
LDV	<number>	Load the given number to ACC
LDD	<address>	Direct addressing. Load the contents of the given address to ACC

Consider the following two cases.

Case 1:
Following step 4 of the fetch stage, the instruction is decoded. The instruction is then executed without further use of the address bus.

Case 2:
Following step 4 of the fetch stage, the instruction is decoded. Once decoded, the address bus must be used again before the execution of the instruction can be completed.

For each instruction below, circle either Case 1 or Case 2 and explain your choice.

(i) `LDV 35`

Case 1 / Case 2

Explanation .....

.....

.....[2]

(ii) `LDD 35`

Case 1 / Case 2

Explanation .....

.....

.....[2]

6 The following are the first few lines of a source program written in a high-level language which is about to be translated by the language compiler.

```
// invoicing program
// program written 21 Oct 2014
DECLARE  i : INTEGER;
DECLARE Customer(40) : STRING;
DECLARE Address: STRING;
CONSTANT DiscountRate = 5;

// start of main program
CALL InitialiseCustomerData
REPEAT
…
…
…
```

(a) During the lexical analysis stage the compiler will use a keyword table and a symbol table.

(i) Describe what information is contained in the keyword table.

.....

.....

.....[2]

(ii) List **three** entries which must be in the keyword table for this program.

.....[1]

(iii) Describe what information is contained in the symbol table.

.....

.....

.....[2]

(iv) List **three** entries which will be entered in the symbol table for this program.

.....[1]

(v) Explain what happens during the lexical analysis stage of compilation. Include how the contents of the keyword table and symbol table are used.

.....

.....

.....

.....

.....

.....

.....

.....

.....[5]

(b) The final stage of compilation is code optimisation.

(i) Explain what is meant by code optimisation.

.....

.....

.....

.....[2]

(ii) Consider **three** assembly language instructions that were given in **Question 4**.

Instruction		Explanation
Op Code	**Operand**	
LDD	<address>	Direct addressing. Load the contents of the given address to ACC
STO	<address>	Copy the contents of ACC to the given address
INC	<register>	Add 1 to the contents of the register (ACC or IX)

Study the assembly language code below.

200	LDD 151
201	INC ACC
202	STO 151
203	LDD 151
204	INC ACC
205	STO 151

One instruction is not needed and could be removed during optimisation.
State the address of this instruction.

Address of instruction .....[1]

7 The function `DateDiff` is documented as follows:

```
FUNCTION DateDiff(Date1 : DATE, Date2 : DATE, OutputFlag : CHAR) RETURNS INTEGER
```

The function assumes `Date1` is earlier than `Date2`.
The function calculates the difference, in days or as a whole number of months, between `Date1` and `Date2`.

`OutputFlag` takes values:
- `'D'` : the result is computed and returned as a number of days
- `'M'` : the result is computed and returned as a whole number of months

An error is generated for each of the following:
- An unrecognised or missing flag parameter
- A date parameter in the wrong format
- `Date1 >= Date2`

Dates are recognised by the function using the 'hash (#) delimiter'.

What is returned from the following function calls?

(a) `DateDiff(#12/09/2014#, #15/09/2014#, 'D')`

.....[1]

(b) `DateDiff(#21/10/2014#, #19/10/2014#, 'D')`

.....[1]

(c) `DateDiff(#30/07/2012#, #30/09/2012#, 'M')`

.....[1]

(d) `DateDiff("12/09/2014", "15/09/2014", 'D')`

.....[1]

(e) `DateDiff(#14/01/2014#, #17/01/2014#)`

.....[1]

(f) High-level programming languages have two types of function. These are **built-in** and **user-defined**.
Explain the difference between them. You may give an example from your practical experience for a built-in function.

.....

.....

.....

.....[2]