CAMBRIDGE INTERNATIONAL EXAMINATIONS

GCE Advanced Level

MARK SCHEME for the October/November 2012 series

9691 COMPUTING

9691/32 Paper 3 (Written Paper), maximum raw mark 90

This mark scheme is published as an aid to teachers and candidates, to indicate the requirements of the examination. It shows the basis on which Examiners were instructed to award marks. It does not indicate the details of the discussions that took place at an Examiners' meeting before marking began, which would have considered the acceptability of alternative answers.

Mark schemes should be read in conjunction with the question paper and the Principal Examiner Report for Teachers.

Cambridge will not enter into discussions about these mark schemes.

Cambridge is publishing the mark schemes for the October/November 2012 series for most IGCSE, GCE Advanced Level and Advanced Subsidiary Level components and some Ordinary Level components.

1 (a) (i) An attribute/or combination of attributes
Which (links to/is) the primary key in another table

(ii) the primary key
Matches/links/connects to a foreign key
Primary in X – foreign key in Y [MAX 3]

(b) (i) Two sensible attributes for Employee (and do not relate to the `Course` or `CourseEnrolment` tables)
`CourseCode` shown as the primary key
One sensible non-key attribute for `Course` (and does not relate to the Employee or `CourseEnrolment` tables) [3]

(ii) `EmployeeID, CourseCode`
Primary key of `EmployeeID + CourseCode` [2]

(c) unnecessary/avoids data duplication/repetition
in normalised relational database tables (addresses the issue) [2]

[Total: 12]

2 (a) (i) 26 [1]

(ii) 102 [1]

(iii) Subtraction is treated in the same way as addition
algorithm to perform addition/subtraction is less complex
There is only one representation for zero
All bits have a place value
A larger range of numbers can be represented [MAX 1]

(b) (i) 15/16 // 0.9375 [1]

(ii) +7 [1]

(iii) 120 seen scores full 2

$15/16 \times 2^{+7}$ // method which shifts the bits in the mantissa [1]
correctly evaluated [1]

(iv) Largest: 0111 1111 0111 [2]
Smallest: 0000 0001 1000 [2]

[Total: 11]

3 (a) (i) The contents of the Memory Data Register are copied to the Current Instruction Register

(ii) *Data Bus:*
copies the contents of the address in MAR to MDR [1]

(iii) *Address bus*
Loaded/carries/transfers contents of MAR [1]

(b) (i) Assembly language is easier/easy to learn//write // program // Understand
Less likely to make coding errors // easier/easy to debug
mnemonics used (give a clue to the nature of the instruction)
Labels can be used to represent addresses [MAX 2]

(ii) translates assembly language instructions into machine code
produces an object/executable file
(checks the syntax of each instruction) reports errors
constructs a symbol table of addresses
converts all symbolic addresses (into absolute/relative addresses)
op codes are looked up from a table
converts macros to a set of instructions // converts constants [MAX 3]

(c) (i) *Direct addressing*
The operand ....
is the actual memory address to be used

e.g. LD 1987 means copy the contents of address 1987 to the Accumulator register [2]

(ii) *Relative addressing*
The operand is an 'offset'
From the address of the current instruction
added to the current PC contents [MAX 2]

[Total: 12]

4 **(a)** Compiler ...
Reports errors in the source code
Produces the executable code/object file/machine code /low-level code [M

(b) *Benefits of a compiler ...*
The program will execute faster
Execution does not require the presence of any translator software
Once compiled the process allows for easy distribution of the executable file(s)
Difficult to reverse engineer the final code [MAX 2]

(c) *Code optimisation*
Possible process which follows the lexical analysis / syntax analysis/ code generation stages
Produce code which executes faster than that produced by the translator software
Produce code which takes up less memory when executed // reduces the amount of program code [3]

(d) **(i)** *Linker software*
needed when the programmer has developed program libraries (which can then be used by many applications)
links segments of code/modules [1]

(ii) *Loader*
loads object/executable code into (main) memory
many are re-locatable loaders [1]

[Total: 9]

5 **(a)** The last item added to the stack will be the first item to leave

(b)
```
TopOfStack = -1
TopOfStack
TopOfStack - 1
```
[1]
[1]
[1]

(c) **(i)** *Application of a stack*
any valid application e.g.
- For the conversion/evaluation of a reverse Polish expression [1]
- Interrupt handling [1]
- procedure calling [1]
- interrupt handling [1]

(ii) Two marks for a clear explanation

E.g. for procedure calling
- Every time a new call is made [1]
- The return address must be stored [1]
- Return addresses are recalled in the order 'last one stored will be the first to be recalled' [1]

[MAX 2]

E.g. for interrupt handling
- Save the register contents / PC contents [1]
- Save the return address
- Retrieve the return address from the stack [1]

[Total 7]

6 (a) Strategies used:

When a job terminates look at the available 'holes' and load next job //
When a job terminates, move all other jobs to create one large hole

Look for the most suitable job from the 'wait list' [MAX 2]

OR
Segmentation
Program is divided into segments of variable size / logical units
Not all the program needs to be loaded at start-up *(once only …)* [MAX 2]

OR
Paging
The program is divided into a number of pages
Pages are a fixed size
The main memory is divided into a number of page frames (of the same size)
Pages continually swapped in/out of memory as required
Not all the program needs to be loaded at start-up *(once only …)* [MAX 2]

OR
Virtual memory
Space on the secondary storage provides addition space which behaves as main memory

OR
Partitioning
Memory divided into fixed areas
Each partition is used for a particular job [MAX 4]

(b) *Job scheduling*
Jobs are allocated a priority [1]
The job with the highest priority gets next use of the processing [1]

Job with the shortest run-time / anything reasonable [1]
Get highest priority [1]
Note: there must be some indication of <u>what</u> is 'shortest' [1]

Round robin [1]
Give each job a time slice of processor time [1]
For each job in the 'ready' state [1]

MAX 3 per strategy [MAX 4]

(c) (i) a signal from some device/program
to indicate that some event has occurred
the source is seeking the attention of the processor [M

(ii) *Example of hardware generated*

Printer
to inform the processor it is out of paper/paper jam/ or similar [2]

the 'Reset' button
actioned by the user [2]

keyboard
to indicate data has been entered and requires saving // key pressed [2]

mouse
e.g. click which will result in a (say) a refresh of the screen [2]

system clock
timer signal [2]

software/program
a 'division by zero' error / file not found / anything reasonable [2]

Mark as 2 + 2 [MAX 4]

[Total: 14]

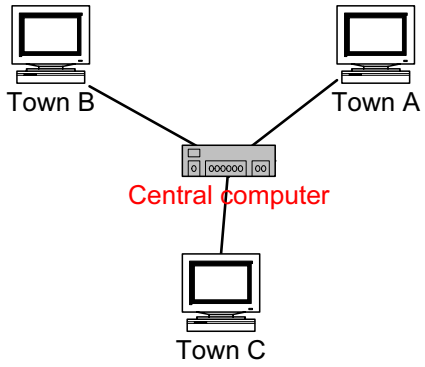
7 **(a)** Two different media
Copper wire
Many different variations – coaxial – twisted pair

Optic fibre cabling
Many signals can be sent on a single fibre
High bandwidth possible
Uses light // Data travels at the speed of light

Radio/Microwave signals
Wireless communication
Bluetooth
Infra-red communication [4]

(b) **(i)** WAN is a collection of computers or networks which are connected
Over a wide geographical area // by example e.g. the Internet [MAX 2]

(ii)



three (A. four) cable runs from a central computer
three computers labelled Town A, Town B and Town C
in star topology

Central computer/Hub/Head Office [3]

(iii) each town uses a separate communication path
different media can be used for each communication link
different speeds can be used for each communication link
if one communication link/terminal/computer fails - other towns are not affected
new node can easily be connected [MAX 2]

[Total: 11]

8 (a) (i) the name given to a function

(ii) `CharacterCount`

(iii) Value(s)/variable(s) which are <u>given/passed</u> (each time the function is called) [1]

(iv) `ThisChar` [1]
`ThisString` [1]

(b) (i) 2 [1]

(ii) 0 [1]

(iii) Error [1]

[Total: 8]

9 (a) (i) `continent(south_america)`
`country(peru, south_america)` [2]

(b) `india, china` [1]

(c) `In_same_continent(`**`Country1, Country2`**`)`

`IF country(`**`Country1,X`**`) AND country(`**`Country2,X`**`)`

Mark as follows:

`country` <u>clause</u> used twice		[1]
`AND` operator between two `country` clauses	(must be caps)	[1]
variables `Country1` and `Country2`	(must be caps)	[1]
Use of a common new variable e.g. `X`		[1]

[MAX 3]

[Total: 6]