CAMBRIDGE INTERNATIONAL EXAMINATIONS

Cambridge International Advanced Level

MARK SCHEME for the October/November 2014 series

9691 COMPUTING

9691/32 Paper 3 (Computing), 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 2014 series for most Cambridge IGCSE®, Cambridge International A and AS Level components and some Cambridge O Level components.

CAMBRIDGE
International Examinations

1 (a) (i) `x y - 4 /` **[1]**

(ii) `3 2 x 7 / +` (1) `*`

Or

`2 x 7 / +` (1) `3 *` **[2]**

(b) (i) `4 * (a + b + c + d + e)`
Accept Omission of the *
Extra brackets as long as the evaluation is correct **[1]**

(ii) `(y^2 + z^3) / 5`
or (1)
Accept `(y^2 + z^3) / 5` scores 1 only **[2]**

(c) (i) Last item added is the first to leave // first add will be the last to leave
Last in – First out // First in – Last out
NE LIFO **[1]**

(ii) Storing return addresses for procedure/function calls

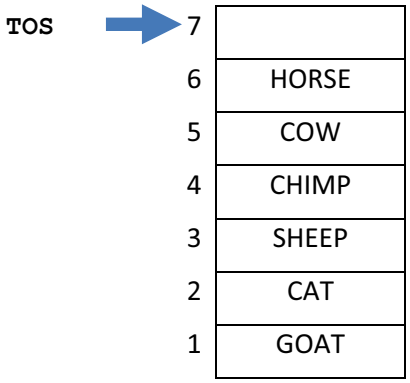
Software focussed
Dealing with the 'Undo' feature in a software application
Printing the pages from a document in reverse order **[1]**

(d) (i) First item added will be the first item to leave //
First in – First out
NE FIFO **[1]**

(ii) Storage of characters codes in a keyboard/printer buffer
Accept buffering
Organisation of spooler jobs in a print spooler
(High-level) scheduling (in a multiprogramming OS) **[Max 1]**

(e) (i)

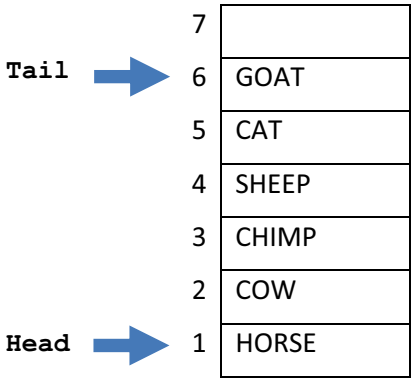
Stack-A



TOS points to 7
6 data items in correct order **[2]**

(ii)

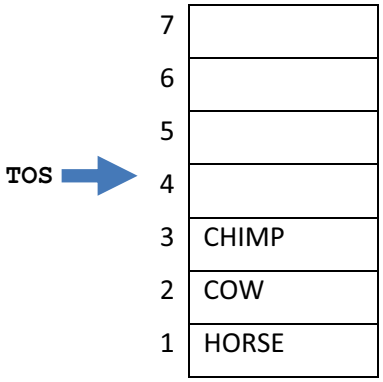
Queue-B



Six values in correct positions 1
Tail = 6 1
Head = 1 1 3

(iii)

Stack-A



TOS points to 4 1
3 data items in correct order 1 **[2]**

(iv) Reverse the order of items on Stack-A **[1]**

2 (a) (i) Pages are managed using a page-map / management table (PMT)
Existing page(s) will be swapped out
Following a particular strategy for deciding which ones
The page containing the (printing) code required is swapped in

Accept for 1 mark only - a description of pages being 'swapped' **[Max 3]**

(b) (i) Round robin
'priority' which is well explained and clear
e.g. Anticipated shortest time to complete

Refuse Priority for either CPU bound or I/O bound **[Max 2]**

(ii) *Processor bound*
Continuously using the CPU // spends very little time doing I/O
1

processing of 3-D graphics //Simulation//weather forecasting processing //
Refuse 'mathematical calculations' 1

I/O Bound
Continuously doing I/O // needs very little CPU time 1

File update // Processing the company payroll (where a lot of output is required)
1 **[4]**

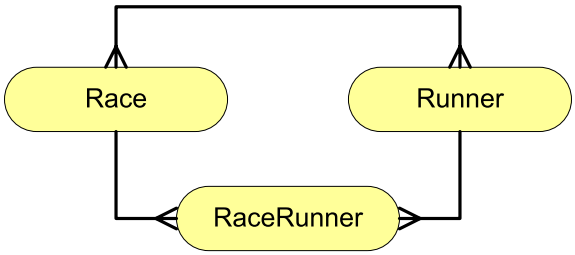
(c) CF / FC
EB / BE
DA / AD for MAX 2

correct sequence 1
(conditional on the 3rd mark …) CF matched with DA // FC matched with AD 1 **[4]**

3 (a) (i)

RaceRunner(RaceDate, RunnerID) **[2]**

(ii)



2 X correct relationships **[2]**

(b) (i) *Not in 2 NF*

RaceRunner // 3 **[1]**

RunnerName is only dependant on knowing part of the PK (i.e. the RunnerID) // there is a non-key attribute which is dependent on only one of the PK attributes **[1]**

RaceRunner(RaceDate, RunnerID FinishingPosition) **[1]**

All correct ..

(ii) *Not in 3NF*

Race // 2 **[1]**

Since there are dependent non-key attributes // ClubSecName and ClubTown are both dependent on ClubName **[1]**

Re-design

Race(RaceDate, RaceDistance, ClubName) **[1]**

New table Club … **[1]**

Club(ClubName, ClubTown, ClubSecretaryName) **[1]**

(c) (i)

```
SELECT RunnerID                                    [1]
FROM RaceRunner                                    [1]
WHERE RaceDate = #26/11/2014#                      [1]
```

(ii)

```
UPDATE RaceRunner                                  [1]
SET FinishingPosition = 2 // 2nd (place)           [1]
WHERE RaceDate = #26/11/2014# AND RunnerID = 8816  [1]
```

4 (a) (i) 256 **[1]**

(ii) Load into the ACC
(The number) 193 // 11000001 **[2]**

(iii) Fewer digits to write // less chance of an error in writing the code // easy conversion to/from a binary code **[1]**

(iv) `05C1 hex` **[1]**

(v) `JPE 204`

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

Op code 1
Operand 1 **[2]**

(vi) True
`OUTCH` // `IN` // `END` // or using a good explanation (only) of either **[2]**

(b)

ACC	Address 450	OUTPUT
65		A
500		
501	501	
74		J
501		
502	502	
65		A
502		
503	503	
90		Z
503		
504	504	
32		

[5]

5 (a) Takes as input a source program
Process identifies errors in the source code
Produces an executable file // object code // machine code
Translation software are not needed at run-time
Use lookup tables/symbol tables **[Max 2]**

Refuse 'in one go' / 'all at once'

(b)

F	F
H	H
B	B
C	C
A	A

Mark as follows:

Look for F – H – B – C – A for full five marks

Or

Not used is D - Run the assembler with the executable code 1

F at the start 1
HB 1
CA 1
Correct sequence of these three blocks 1 **[5]**

(c) (i) Interpreters usually provide better diagnostics / easier to debug /or by example
Note: Must hint at a comparison with a compiler so Refuse 'easy to debug' 1

Using an interpreter will allow some parts of the program (only) to be tested and run // without all the program code being available 1

Fits with the strategy of a modular approach (to program design and coding). 1 **[Max 2]**

(ii) The interpreter software must always be present whenever we attempt to execute the program // no final executable file is produced 1

The interpreter must interpret and execute each statement every time the program is run 1

The program will execute slower (compared to compiled code) 1 **[Max 1]**

6 (a) 1. The (contents of) the program counter/PC are copied to the Memory Address Register
Refuse 'instruction' stated as 'the contents of'
2. The contents of the Program Counter are incremented
3. Identify the address in the Memory Address Register. Go to this address and copy its contents to the Memory Data Register
4. The (contents of) the Memory Data Register are copied to the Current Instruction Register **[4]**

(b) (i) Control bus **[1]**

(ii) read / write
interrupt
reset
Clock signal
Bus request / bus grant **[Max 1]**

(c) (i) Case 2 1

The address in CIR must be loaded to the MAR / address bus 1 **[Max 2]**
The data value must be retrieved from this address / address 78 1

(ii) Case 1 1

The operand is a register // the register is part of the CPU (i.e. not in memory) // it is using only the Accumulator 1 **[Max 2]**
the address bus is not used // there is no call to memory 1

7 (a) `Mary Kelly` **[1]**

(b) `1X0X` **[1]**

(c) `Ajaz ew` **[1]**

(d) Error **[1]**

(e) `white box TESTING` **[1]**

(f) Built-in functions are those provided (as a part of the programming language) // accept by example 1
User defined functions are designed and coded by the programmer 1 **[2]**