**CAMBRIDGE INTERNATIONAL EXAMINATIONS**

**Cambridge International Advanced Level**

# MARK SCHEME for the October/November 2015 series

## 9608 COMPUTER SCIENCE

**9608/31** Paper 3 (Written Paper), maximum raw mark 75

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 2015 series for most Cambridge IGCSE®, Cambridge International A and AS Level components and some Cambridge O Level components.

**1 (a) (i)** 00101000 00000011
=0.0101 × 2 ↑3 [1]
=10.1 [1]
=2.5 [1]

**(ii)** For a positive number (mantissa starts with a zero) [1]
bit after binary point (second bit from left) should be a one [1]

**(iii)** 00101000 00000011
= 01010000 00000010 [1+1]

**(b) (i)** 01111111 0111111 [1+1]

**(ii)** 01000000 1000000 [1+1]

**(iii)** number will become too large to represent [1]
which will result in overflow [1]

**(c)** Any point 1 mark

0.1 cannot be represented exactly in binary
0.1 represented here by a value just less than 0.1
the loop keeps adding this approximate value to counter
until all accumulated small differences become significant enough to be seen

[max 3]

**2 (a)**

| Symbol | Token | |
|---|---|---|
| | Value | Type |
| Counter | 60 | variable |
| 1.5 | 61 | constant |
| Num1 | 62 | variable [1] |
| 5.0 | 63 | constant [1+1] |

**(b)**

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 60 | 01 | 61 | 51 | 62 | 4A | 60 | 03 | 62 | 4B | 62 | 01 | 62 | 02 | 63 | 4C |

[1+1]

**(c) (i)** Code optimisation [1]

**(ii)** LDD 234
ADD 235 [1]
ADD 236
STO 233 [1]

1 mark for first 2 lines, 1 mark for last 2 lines, with no other lines added

**(iii)** Code has fewer instructions / occupies less space in memory when executed [1]
minimises execution time of code // code will execute faster [1]

**3 (a)** Any point 1 mark

sender's IP address
receiver's IP address
packet sequence number
checksum

[Max 2]

**(b)** Any point 1 mark

email has been split up into packets
packet has destination address
packets pass through many different routers in journey
packets don't take same route
routers use IP addresses
packets reassembled at destination to rebuild email

[Max 3]

**(c)** Any point 1 mark

email message is only read when all of it is received
time delays due to lost / delayed packets not significant
so sending different packets by different routes is not issue / is efficient
packets arriving out of order not an issue
no requirement for a continuous circuit (circuit switching)

[Max 2]

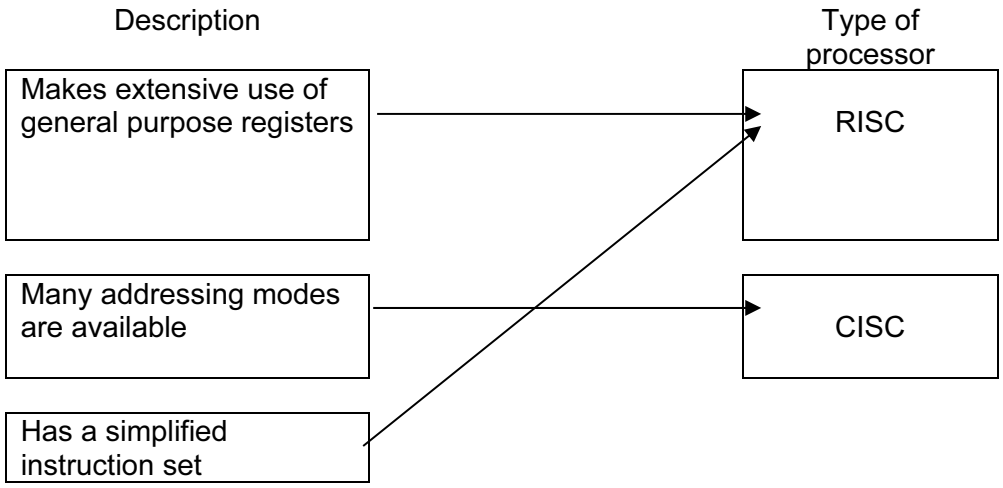
**(d)** Circuit switching [1]

**(e)** e.g. real-time video / video conferencing [1]

Any point 1 mark

circuit made available is dedicated to this communication stream
full bandwidth available / no sharing
no lost packets
guaranteed quality of service [Max 2]

**4 (a)**




1 mark for correct arrow from each description

[3]

**(b) (i)**

Time Interval

| stage | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| Fetch instruction | A | B | C | | | | | | |
| Decode instruction | | A | B | C | | | | | |
| Execute instruction | | | A | B | C | | | | |
| Access operand in memory | | | | A | B | C | | | |
| Write result to register | | | | | A | B | C | | |

Completing the As (1 Mark)

B in column 2, Row 1 (1 Mark)

Remainder completed (1 Mark)

[3]

**(ii)** With pipelining no of cycles = 7 [1]
Without pipelining no of cycles = 3 * 5 = 15 [1]
No of cycles saved = 8 [1]

**5 (a) (i)** $\overline{A}.B.C +$ [1]

$A.B.\overline{C}$ [1]

$+$

$A.B.C$ [1]

**(ii)**

**AB**

| C | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| **0** | 0 | 0 | 1 | 0 |
| **1** | 0 | 1 | 1 | 0 |

[1]

**(iii)**

**AB**

| C | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| **0** | 0 | 0 | 1 | 0 |
| **1** | 0 | 1 | 1 | 0 |

1 mark for each loop

Allow f.t. from **(ii)**

[2]

**(iv)** $X =$

$A.B$ [1]

$+ B.C$ [1]

Allow f.t. from **(iii)**

**(b) (i)**

| CD \ AB | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 1 | 1 | 0 |
| 01 | 0 | 0 | 0 | 0 |
| 11 | 0 | 0 | 1 | 0 |
| 10 | 0 | 1 | 1 | 0 |

1 mark row headings

1 mark column headings

1 mark per 2 correct rows (based on headings)

[4]

**(ii)**

| CD \ AB | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 1 | 1 | 0 |
| 01 | 0 | 0 | 0 | 0 |
| 11 | 0 | 0 | 1 | 0 |
| 10 | 0 | 1 | 1 | 0 |

1 mark for loop with two 1s

1 mark for looping the four 1s

[2]

**(iii)** $X = B.\overline{D}$ [1]

$+ A.B.C$ [1]

**6** **(a)** A program is the written code ("static") [1]
A process is the executing code ("dynamic") [1]

**(b)** **running, ready:**
when process is executing it is allocated a time slice (running state) // process is allocated time on processor [1]
when time slice completed process / interrupt occurs can no longer use processor even though it is capable of further processing (ready state) [1]

**ready, running:**
process is capable of using processor (ready state) [1]
OS allocates processor to process so that process can execute (running state) [1]

**running, blocked:**
process is executing (running state) when it needs to perform I / O operation [1]
placed in blocked state – until I / O operation completed [1]

**(c)** when I / O operation completed for process in blocked state [1]
process put in ready state [1]
OS decides which process to allocate to processor from the ready queue [1]

**(d)** **high-level scheduler:**
decides which processes are to be loaded from backing store [1]
into memory / ready queue [1]