5.1 Number systems Mark Scheme

Q1.

(a) **All marks AO1 (understanding)**

15/23: Rational, Real;
108: Natural, Integer, Rational, Real;
R. answers in which additional lozenges are shaded

2

(b) **1 mark AO1 (knowledge) and 1 mark AO1 (understanding)**

What is – 1 mark (AO1(knowledge)):
Shows order / position / rank / place;
Use in array – 1 mark (AO1(understanding)):
The ordinal numbers would represent the position / index / location of the values in the array;

2

[4]

Q2.

(a) **Mark is for AO1 (understanding)**

Any number from the set of natural numbers;
{0,1,2,3,....}

1

(b) **Mark is for AO1 (understanding)**

Any number from the set of irrational numbers;
Examples: square root of 2, pi, Euler's number (e)

1

[2]

Q3.

(a) **All marks AO1 (understanding)**

1 mark per correct response:

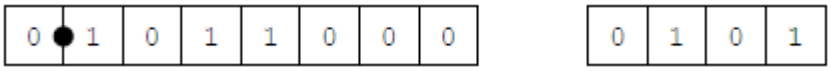
Value description	Correct letter (A-D)
A positive normalised value.	A
The most negative value that can be represented.	C
A value that is not valid in the representation because it is not normalised.	B

If a letter is used more than once then mark as correct in the position where it

is correct (if any).

3

(b) **All marks AO2 (apply)**



Mantissa Exponent

1 method mark for either:
- showing correct value of both mantissa and exponent in denary (Mantissa = 0.6875 / / 11 / 16, Exponent = 5)
- showing binary point shifted 5 places to right in binary number
- indicating that final answer calculated using answer = mantissa x $2^{exponent}$

1 mark for correct answer

Answer = 22

If answer is correct and some working has been shown, award two marks, even if working would not have gained credit on its own.

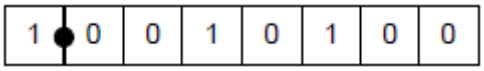
2

(c) **All marks AO2 (apply)**

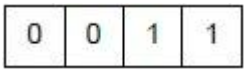
2 marks for working:

Correct representation of 6.75 in fixed point binary:
110.11; **A** leading 0s.
Correct representation of -6.75 in two's complement fixed point binary:
1001.01; **A** leading 1s.
Showing the correct value of the exponent in denary (3) or binary (11) / / showing the binary point being shifted 3 places;
Max 2

1 mark for correct mantissa and exponent together:



Mantissa



Exponent

If answer is correct and some working has been shown, award three marks, even if working would not have gained credit on its own.

Working marks can be awarded for work seen in the final answer eg correct exponent.

3

(d) **All marks AO1 (understanding)**

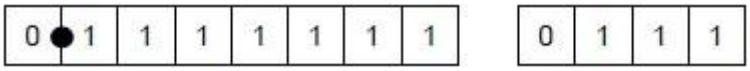
1 mark: Reduced precision;
1 mark: Increased range; **A** can represent larger / smaller numbers

2

[10]

Q4.

(a)

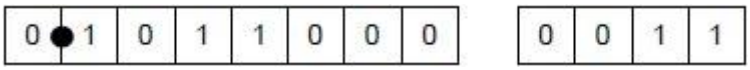


Mantissa Exponent

1 mark for correct mantissa
1 mark for correct exponent

2

(b)



Mantissa Exponent

1 method mark for either:
- showing correct value of both mantissa and exponent in denary (Mantissa = 0.6875 / / 11 / 16, Exponent = 3)
- showing binary point shifted 3 places to right in binary number
- indicating that final answer calculated using answer = mantissa x $2^{exponent}$

1 mark for correct answer

Answer = 5 ½ / / 5.5

*If answer is correct and some working has been shown, award **2 marks**, even if working would not have gained credit on its own.*

2

(c)

Mantissa Exponent

1 method mark for either:
- showing correct value of both mantissa and exponent in denary (Mantissa = -0.625 / / -5 / 8, Exponent = -4)
- showing binary point shifted 4 places to left in binary number
- indicating that final answer calculated using answer = mantissa x $2^{exponent}$

1 mark for correct answer

Answer = -5 / 128,-0.0390625 **A** rounded to at least 2dp

If answer is correct and some working has been

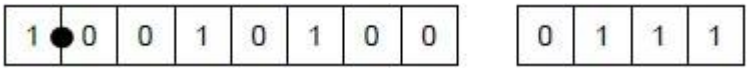
*shown, award **2 marks**, even if working would not have gained credit on its own.*

2

(d) **2 marks** for working:
Correct representation of 108 in binary: 1101100; **A** any number of preceding 0s or succeeding 0s after a binary point
Correct representation of -108: 10010100; **A** any number of preceding 1s
Showing the correct value of the exponent in denary (7) or binary (0111) / / showing the binary point being shifted 7 places;
Showing the correct value of the mantissa in floating point binary: 1.0010100;

MAX 2

1 mark for correct mantissa and exponent together:



Mantissa Exponent

*If answer is correct and some working has been shown, award **3 marks**, even if working would not have gained credit on its own.*
Marks for working can be awarded in the answer.

3

(e) (i) The results of a calculation; is a number that is too large to store; in the available storage space / number of bits;
Must get the middle point (about the number being too large) to be awarded any marks.
MAX 2

2

(ii)

Operation	**May cause overflow? (Tick One)**
Subtracting a very small number from a large number.	
Dividing a large number by a very small number.	✓

Multiplying a large number by a very small number.	

A alternative symbol which clearly indicates just one box eg cross, Y, Yes
R answers in which more than one row is ticked

1

[12]

Q5.

(a) One mark per correct answer:

Value description	**Correct letter (A-D)**
A negative value.	D;
The smallest positive value that can be represented.	A;
A value that is not valid in the representation because it is not normalised.	C;

If a letter is used more than once then mark as correct in the position that is correct.

3

(b) **1 method mark** for either:

- showing correct value of both mantissa and exponent in denary
- showing binary point shifted 6 places to right in mantissa
- indicating that final answer calculated using answer = mantissa x $2^{exponent}$

Mantissa = 0.625 // 5/8
Exponent = 6

1 mark for correct answer

Answer = 40

If answer is correct and some working has been shown, award two marks, even if working would not have gained credit on its own.

2

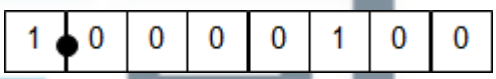
(c) **2 marks** for working:

Correct representation of 7.75 in fixed point binary: 111.11;
A leading and trailing 0s.
Bits flipped: 000.00 // 1000.00; **A** leading 1s
Correct representation of -7.75 in fixed point two's complement: 1000.01;
A leading 1s
Showing the correct value of the exponent in denary (3) or binary (11) // showing the binary point being shifted 3 places;

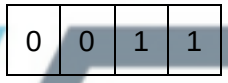
Note: Award both working marks if bit pattern 1.00001 is shown anywhere

Max 2

1 mark for correct mantissa and exponent together



Mantissa



Exponent

If answer is correct and some working has been shown, award three marks, even if working would not have gained credit on its own.

Working marks can be awarded for work seen in the final answer e.g. correct exponent.

3

(d) (i) 0.025 // 6.9-6.875 // 1/40
R -0.025
A award **BOD** mark if correct method has been shown i.e. 6.9-6.875 but candidate has then made an error performing the subtraction operation

1

(ii) 0.003623 // 0.025/6.9 // 1/276
A 0.3623%
A answers rounded to at least two significant figures
A follow-through of incorrect answer to part (d)(i)
A award **BOD** mark if correct method has been shown but candidate has then made an error

performing the division operation
R if shown that incorrect method used e.g. dividing by 6.875, even though this arrives at an answer that is the same when written to 2 significant figures

1

(iii) **Alternative 1:**
Adjust the mantissa;
To use more bits;
A "longer" for "more bits" but **R** "larger", "increase size"
Alternative 2:
Reallocate (one) bit; from the exponent to the mantissa; **A** bits
Alternative 3:
Infer one of the two bits on either side of the binary point (from the other, as they must both be different); use the freed up bit to store one more significant digit in the mantissa // use the freed up bit to represent mantissa more accurately;

2

[12]

Q6.

(a) Correct variable declarations for `Bit, Answer` and `Column;`
I additional variable declarations
`Column` initialised correctly before the start of the loop;
`Answer` initialised correctly before the start of the loop;
`While/Repeat` loop, with syntax allowed by the programming language used,
after the variable initialisations; and correct condition for the termination of the loop;
R `For` loop
A any `While/Repeat` loop with logic corresponding to that in flowchart
(for a loop with a condition at the start accept >=1 or >0 but reject <>0)
Correct prompt `"Enter bit value:"` ;
followed by `Bit` assigned value entered by user;
followed by `Answer` given new value;
R if incorrect value would be calculated [followed by value of `Column` divided by 2;
A multiplying by 0.5
Correct prompt and the assignment statements altering `Bit, Answer` and
`Column` are all within the loop;
After the loop – output message followed by value of `Answer`;
I Case of variable names, player names and output messages
A Minor typos in variable names and output messages
I spacing in prompts
A answers where formatting of decimal values is

included e.g. `Writeln('Decimal value is: ', Answer : 3)`
A initialisation of variables at declaration stage
A no brackets around column * bit

Pascal

```
Program Question;
    Var
         Answer : Integer;
         Column : Integer;
         Bit : Integer;
    Begin
          Answer := 0;
          Column := 8;
          Repeat
              Writeln('Enter bit value: ');
              Readln(Bit);
              Answer := Answer + (Column * Bit);
              Column := Column DIV 2;
         Until Column < 1;
         Writeln('Decimal value is: ', Answer);
         Readln;
    End.
```

VB.NET

```
Sub Main()
   Dim Answer As Integer
   Dim Column As Integer
   Dim Bit As Integer
   Answer = 0
   Column = 8
   Do
        Console.Write("Enter bit value: ")
        Bit = Console.ReadLine
        Answer = Answer + (Column * Bit)
        Column = Column / 2
  Loop Until Column < 1
  Console.Write("Decimal value is: " & Answer)
  Console.ReadLine()
End Sub
```


Alternative Answer

```
Column = Column \ 2
```

VB6

```
Private Sub Form_Load()
   Dim Answer As Integer
   Dim Column As Integer
   Dim Bit As Integer
   Answer = 0
   Column = 8
   Do
        Bit = InputBox("Enter bit value: ")
        Answer = Answer + (Column * Bit)
        Column = Column / 2
  Loop Until Column < 1
  MsgBox ("Decimal value is: " & Answer)
End Sub
```

Alternative Answer

```
Column = Column \ 2
```

Java

```
public class Question {
   AQAConsole console=new AQAConsole();
   public Question(){
         int column;
         int answer;
         int bit;
         answer=0;
         column=8;
         do{
              console.print("Enter bit value: ");
              bit=console.readInteger("");
              answer=answer+(column*bit);
              column=column/2;
         }while(column>=1);
         console.print("Decimal value is: ");
         console.println(answer);
      }
      public static void main(String[] arrays){
          new Question();
      }
}
```

Python 2.6

```
Answer = 0
Bit = 0
Column = 8
while Column >= 1:
   print "Enter bit value: "
   # Accept: Bit = int(raw_input("Enter bit value: "))
   Bit = input()
   Answer = Answer + (Column * Bit)
   Column = Column // 2
print "Decimal value is: ", Answer
# or + str(Answer)
```

Python 3.1

```
Answer = 0
Bit = 0
Column = 8
while Column >= 1:
   print("Enter bit value: ")
   # Accept: Bit = int(input("Enter bit value: "))
   Bit = int(input())
   Answer = Answer + (Column * Bit)
   Column = Column // 2
print("Decimal value is: " + str(Answer))
# or print("Decimal value is: {0}".format(Answer))
```


A. Answer and Bit not declared at start as long as they are spelt correctly and when they are given an initial value that value is of the correct data type

11

(b) ****SCREEN CAPTURE****
Must match code from 16, including prompts on screen capture matching those in code

Mark as follows:
"Enter bit value:" + first user input of 1
‘Enter bit value: ’ + second user input of 1
‘Enter bit value: ’ + third user input of 0
‘Enter bit value: ’ + fourth user input of 1

Value of 13 outputted;

3

(c) 15;

1

(d) 16 // twice as many // double;

1

(e) Design;
A Planning

1

(f) Implementation;

1

[18]

Q7.

(a) 0111 1011;

1

(b) 256 // 2^8;

1

(c) 7;B;

2

(d) Easier for <u>people</u> to read / understand;
(Can be displayed using) fewer digits;
More compact when printed/displayed;
NE Takes up less space
NE More compact

Max 1

[5]

Q8.

(a)

Real number	Yes / No
203.412	**Yes**
–12.87	**No**
12.43E–12	**Yes**

1 mark per correct Yes / No

A other indicators that clearly mean Yes / No e.g. True/False, Tick / Cross.

3

(b) <digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<whole-number> ::= <digit> | <digit> <whole-number>
<integer> ::= <whole-number> | + <whole-number> |
- <whole-number>

1 mark for each correct rule

Alternative for integer (1 mark, accept in either order):
<symbol> ::= + | -
<integer> ::= <whole-number> | <symbol> <whole-number>
A <whole-number> defined with recursion other way around, i.e.
<whole-number> ::= <digit> | <whole-number> <digit>
A non-terminal names e.g. digit not enclosed in <> signs
A spaces in non-terminal names e.g. whole number
A terminal names enclosed in quotation marks e.g. “0”, ‘0’.
A any sensible symbol for assignment e.g. ←, :=, =, :
A ; as end-of-rule marker;
A any type of slash e.g. / for alternatives but **R** "or"
A use of EBNF extensions for repetition and optional terms:
<whole-number> ::= <digit> { <digit> }
<integer > ::= [+ | -] <whole-number> **A** () for [] but **R** {}
R rules that have additional options e.g. more than ten digits
DPT addition of chevrons or other symbols such as brackets to terminal symbols/rules unless they make meaning unclear

3

[6]

Q9.

(a) 3rd (generation);

1

(b) (i) (Program) 2; (Program) 3;

1

(ii) (Program) 1;

1

(c)

	Assembler	Compiler	None
Program 1		✓	
Program 2	✓		
Program 3			✓

If more than one entry per row – look for a single tick
If mixture of X and ticks used – mark (as long as only one tick per row)

3

(d) The interpreter software is resident in memory at the same time as the application program is run;
The interpreter recognises/translates/reads/converts each statement **A** instruction/line;
T/O if added “converts to machine code”
(Syntax) checks the program; line-by-line; if ‘error free’ the statement /line is executed;
The interpreter calls a procedure/code to execute the statement;
When (first) error encountered program execution is halted;

Max 2

(e) The processor/architecture/(hardware) platform is different;
instructions are not the same;

Assembler software is processor/architecture specific;
A Assembler software is 'machine specific'

The computers use a different operating system;
R the 'computer' might be …
R possible bugs in the software

Max 1

[9]

Q10.

(a) 255 / 11111111 / FF / 2^8 – 1 ;

1

(b) (i) 81 ;

1

(ii) 1000 1010 ;

1

(c) 522

1

(d) (i) 100 0010 ;
R leading 0 as 8th bit

1

(ii) The total is calculated before transmission ;
0 or 1 bit is added (to the 7-bit code) ;
So that, the total number of 1 bits must compute to an even number ;
The number of one bits is re-calculated after received ;
An odd number of 1 bits indicates an error ;

Max 2

[7]

Examiner reports

Q1.

(a) This question was well tackled, with over 80% of students achieving some marks and about half achieving both marks.

(b) This question was well answered. Three quarters of students were able to identify that an ordinal number was used to specify the position of an item in a list and the majority of these then went on to explain that in the context of the array the index was an ordinal number. The most common misunderstanding was to believe that an ordinal number referred to the length of an array.

Q4.

As in previous years, candidates demonstrated a very good understanding of floating point representation.

(a) This part was extremely well tackled, with over three quarters of students achieving full marks.

(b) This part was extremely well tackled, with over three quarters of students achieving full marks.

(c) This part was not tackled as well, with less than half of the candidates achieving both marks. Some candidates mistakenly moved the binary point four places to the right instead of the left or miscalculated the exponent to be 12 instead of -4. Others did not know how to deal with the binary point being moved to the left past the sign bit in a negative number and so ended up with a positive number instead of a negative number at the end.

(d) This part of the question was well tackled, with the majority of students achieving full marks and two thirds achieving at least two of the three marks. Those students who did not achieve full marks most commonly went wrong when converting 108 in unsigned binary into -108 in two's complement. Another mistake was to write out the binary place values backwards when working out the bit pattern for 108, ie 1,2,4,8 etc.

(e) The vast majority of candidates achieved at least one mark for this part, correctly recognising that overflow occurred when a value was too large to be stored in the number of bits available. The clearest answers explicitly stated that the number of bits available for storing the number was insufficient, rather than making more vague references to, for example, memory space. Some students failed to achieve a mark because they related the number of bits to the mantissa alone. Around two thirds of candidates were able to identify correctly that the situation which might cause overflow was division for this part.

Q5.

As in previous years, the majority of candidates demonstrated a sound ability to manipulate floating point numbers.

The majority of students achieved full marks for parts (a) and (b).

Part (c) was well answered. Candidates who made mistakes tended to do so because they had incorrectly represented -7.75 as a fixed point number at the start of the process.

For parts (d)(i) and (d)(ii) this was the first time that candidates had been asked to calculate absolute and relative error values and responses to these question parts were mixed. Part (i) was much better answered than part (ii) was, with about two thirds of candidates achieving the mark for the former but only one third for the latter. The most common mistakes in part (i) were to work out the relative error instead of the absolute error, or to just divide 6.9 by 6.875 (or vice-versa). Many candidates simply did not write a response to part (ii) and of those who did, a lot simply rewrote their answers to part (i), presumably in the hope that the value calculated would get them one of the two marks. A few candidates made the mistake of dividing by 6.875 rather than 6.9.

Question (d)(iii) asked about improving the accuracy of the representation and was very well answered with the majority of candidates recognising that one way to do this would be to include more bits in the mantissa. A common error was to state that mantissa needed to be “larger" or “increased" which was not creditworthy, as these responses suggested that the magnitude of the mantissa would need to be changed rather than the number of bits used to represent it.

Q6.

For the first time a flowchart was used to represent an algorithm in a COMP1 exam. There was no increase in difficulty resulting from this and the standard of answers was the same as seen in the previous year.

Some students did not follow the algorithm given and instead developed their own program to convert binary to denary. This resulted in them not getting many marks as they had not answered the question.

Students using VB6 tended to get lower marks on this question than those using the other languages available for COMP1. This was partly due to not providing the correct evidence for the testing (screen captures needed to show the data entered for the test as well as the result of the test), although many students using VB6 also seemed to have weaker programming skills.

Students need to be aware that an algorithm is not the same as a program and that simply copying the algorithm into their development environment will not result in a working program in any of the COMP1 programming languages – the pseudo–code/flowchart needs to be adapted to match the syntax of the programming language they are using. As in previous years, a number of students simply copied parts of the algorithm into their program code eg trying to use a keyword of OUTPUT. These appeared to be less able students who generally struggled on the Section D programming as well. The vast majority of students were able to convert the algorithm successfully into working program code and the marks obtained on this question were virtually identical to those achieved on Section B on the 2011 COMP1 exam.

Q7.

Most candidates obtained the mark for part (a). Those who did not get the mark often used 7 bits instead of 8 – 1111011 – (missing out the 0 on the LHS) or miscounted the number of 1s to use – 0111011. Part (b) was generally well-answered. The most common incorrect answers were 255 (the highest decimal value that can be represented using 8 bits) and 128 (the number of different values that can be represented using 7-bit binary). This suggests that candidates are recognising that they need to do 2^8, but often then get confused about when they need to take a 1 away – getting muddled finding the highest value with finding out the number of values that can be represented. Some candidates are taking 1 away from 2^8 and some are taking 1 away from 8 and giving an answer of 2^7. A larger number of candidates got the correct answer to the hexadecimal question this year, but a significant proportion did not understand what hexadecimal is and made either

no attempt or gave an answer that was not in hexadecimal. Answers for part (d) often lacked precision and did not convey the idea that it is easier for a person to read the hexadecimal equivalent of a binary bit pattern. A common misconception is that hexadecimal values will use less storage space.

Q8.

Part (a): The vast majority of candidates scored full marks for this question part, correctly identifying whether or not the numbers were syntactically valid.

Part (b): There were some good responses to this question part, although some candidates clearly found it quite difficult. This is the first question that has been asked on this topic so, on this occasion, we were lenient when faced with minor syntactical errors. A small number of candidates confused Backus-Naur Form with regular expressions.

Q9.

(a)(b)

Candidates generally scored well on these parts which was encouraging. At this level is it likely that candidates will have had little or no practical experience of low level programming. Therefore it was sufficient that the candidate was able to recognise the various generations and appreciate likely applications for which each would be used to score the marks.

(c) Well answered.

(d) This question was a variation from those in previous papers where candidates had been asked to compare compiler and interpreter software. There were many different points which the candidates could have made to secure the two marks. Most common correct answers suggested 'the interpreter translates each statement' – although answers which went on to suggest 'into machine code' were considered to have talked themselves out of this mark - the concept that it does the translation 'line by line'. The mark scheme was considered generous but, despite this, very few candidates scored the maximum 2 marks.

Candidates must read the question. There were often statements such as 'each HHL statement translates to several LLL statements' which suggests knowledge but did not answer this question.

(e) This question was asked for the first time, but something more than 'because they are not compatible' was looked for. The assembler software is specific to a particular processor (architecture) is what was wanted.

Q10.

Parts (a) (b) and (c) were often poorly answered which is at odds with the general level of answers seen on previous papers. Was it because the computations were put into a context that students were unable to identify with?

(d) (i) Some candidates did not read the rubric and wrongly gave an 8-bit code.

(ii) Again the candidates' communication skills often let them down. Their answer was often sufficient to suggest to the examiner that the candidate probably did know how parity worked but they were unable to express the key parts of the process – there were many points the candidate could have described to secure the two available marks.