A

A-level
COMPUTER SCIENCE
Paper 2
7517/2

Insert

FIGURE 5 and source information for use in answering Question 7

TABLE 1 and FIGURE 7 for use in answering Question 9

FIGURE 8 and source information for use in answering Question 12

[Turn over]

0 7

A network of zoos uses a relational database system to store information about the animals that they have so that they can be matched up with animals at other zoos in a breeding programme.

FIGURE 5 shows the structure of the relations in the database.

FIGURE 5

Zoo(ZooName, Town, Country)

AnimalLocation(AnimalID, ZooName, DateArrived, DateLeft)

Animal(AnimalID, IndividualName, Species, DateOfBirth, Sex)

Match(AnimalFemaleID, AnimalMaleID, DateOfMatch, Successful)

- **The Zoo relation stores details of the zoos that participate in the breeding programme. Each zoo is uniquely identified by its ZooName.**
- **The AnimalLocation relation identifies which zoos each animal has lived at. The zoo that the animal is currently at can be identified because the DateLeft attribute is set to 01/01/0001 to indicate that the animal has not left.**

- **The Animal relation stores details of the individual animals that are available to be matched with other animals for breeding. Each animal is identified by a unique number, the AnimalID. The individual name of the animal (eg 'Timothy') is also stored, together with the species of the animal (eg 'Red Panda'), its date of birth and its sex ('Male' or 'Female').**

- **The Match relation stores details of matches that have been made. The attributes AnimalFemaleID and AnimalMaleID refer to the AnimalID values of the two matched animals in the Animal relation.**

[Turn over]

TABLE 1

Standard AQA assembly language instruction set

LDR Rd, <memory ref>	**Load the value stored in the memory location specified by** <memory ref> **into register** d**.**
STR Rd, <memory ref>	**Store the value that is in register** d **into the memory location specified by** <memory ref>**.**
ADD Rd, Rn, <operand2>	**Add the value specified in** <operand2> **to the value in register** n **and store the result in register** d**.**
SUB Rd, Rn, <operand2>	**Subtract the value specified by** <operand2> **from the value in register** n **and store the result in register** d**.**
MOV Rd, <operand2>	**Copy the value specified by** <operand2> **into register** d**.**
CMP Rn, <operand2>	**Compare the value stored in register** n **with the value specified by** <operand2>**.**
B <label>	**Always branch to the instruction at position** <label> **in the program.**
B <condition> <label>	**Branch to the instruction at position** <label> **if the last comparison met the criterion specified by** <condition>**.** **Possible values for** <condition> **and their meanings are:** EQ**: equal to** NE**: not equal to** GT**: greater than** LT**: less than**

`AND Rd, Rn, <operand2>`	**Perform a bitwise logical AND operation between the value in register `n` and the value specified by `<operand2>` and store the result in register `d`.**
`ORR Rd, Rn, <operand2>`	**Perform a bitwise logical OR operation between the value in register `n` and the value specified by `<operand2>` and store the result in register `d`.**
`EOR Rd, Rn, <operand2>`	**Perform a bitwise logical XOR (exclusive or) operation between the value in register `n` and the value specified by `<operand2>` and store the result in register `d`.**
`MVN Rd, <operand2>`	**Perform a bitwise logical NOT operation on the value specified by `<operand2>` and store the result in register `d`.**
`LSL Rd, Rn, <operand2>`	**Logically shift left the value stored in register `n` by the number of bits specified by `<operand2>` and store the result in register `d`.**
`LSR Rd, Rn, <operand2>`	**Logically shift right the value stored in register `n` by the number of bits specified by `<operand2>` and store the result in register `d`.**
`HALT`	**Stops the execution of the program.**

BLANK PAGE

LABELS: A label is placed in the code by writing an identifier followed by a colon (:). To refer to a label, the identifier of the label is placed after the branch instruction.

Interpretation of `<operand2>`

`<operand2>` **can be interpreted in two different ways, depending on whether the first character is a # or an R:**

- `#` **– use the decimal value specified after the** `#`**, eg** `#25` **means use the decimal value 25**
- `Rm` **– use the value stored in register** `m`**, eg** `R6` **means use the value stored in register 6**

The available general-purpose registers that the programmer can use are numbered 0–12

[Turn over]

FIGURE 7

```
  LDR R0, 120
  LDR R1, 121
  MOV R3, #0
loop:
  CMP R1, #0
  BEQ exit
  AND R2, R1, #1
  CMP R2, #0
  BEQ skip
  ADD R3, R3, R0
skip:
  LSL R0, R0, #1
  LSR R1, R1, #1
  B loop
exit:
  STR R3, 122
  HALT
```

BLANK PAGE

[Turn over]

1 2

In a functional programming language, six functions named `fu`, `fv`, `fw`, `fx`, `fy` **and** `fz` **and a list of temperatures in Fahrenheit named** `temps` **are defined as shown in FIGURE 8.**

FIGURE 8

```
temps = [50, 68, 95, 86]

fu a = (a - 32) * 5 / 9

fv b = map fu b

fw [] = 0
fw (x:xs) = 1 + fw (xs)

fx [] = 0
fx (x:xs) = x + fx (xs)

fy c = fx (c) / fw (c)

fz d = fy (fv (d))
```

A temperature can be converted from degrees Fahrenheit to degrees centigrade using the following method:

$$\text{centigrade} = (\text{Fahrenheit} - 32) \times \frac{5}{9}$$

For example, 59 degrees Fahrenheit is equivalent to 15 degrees centigrade.

In the functions `fw` **and** `fx`**:**

- `[]` **is the empty list**
- `(x:xs)` **lets the function definition refer to the head of the list as** `x` **and the tail as** `xs`.

END OF SOURCE MATERIAL

BLANK PAGE