A

A-level

COMPUTER SCIENCE

Paper 2

7517/2

Insert

FIGURE 5 for use in answering Question 3.

FIGURE 6 for use in answering Question 5.

FIGURE 7 for use in answering Question 7.

FIGURE 11 for use in answering Question 11.

FIGURE 12 for use in answering Question 12.

TABLE 3 for use in answering Question 12.

TABLE 4 for use in answering Question 12.

[Turn over]

FIGURE 5

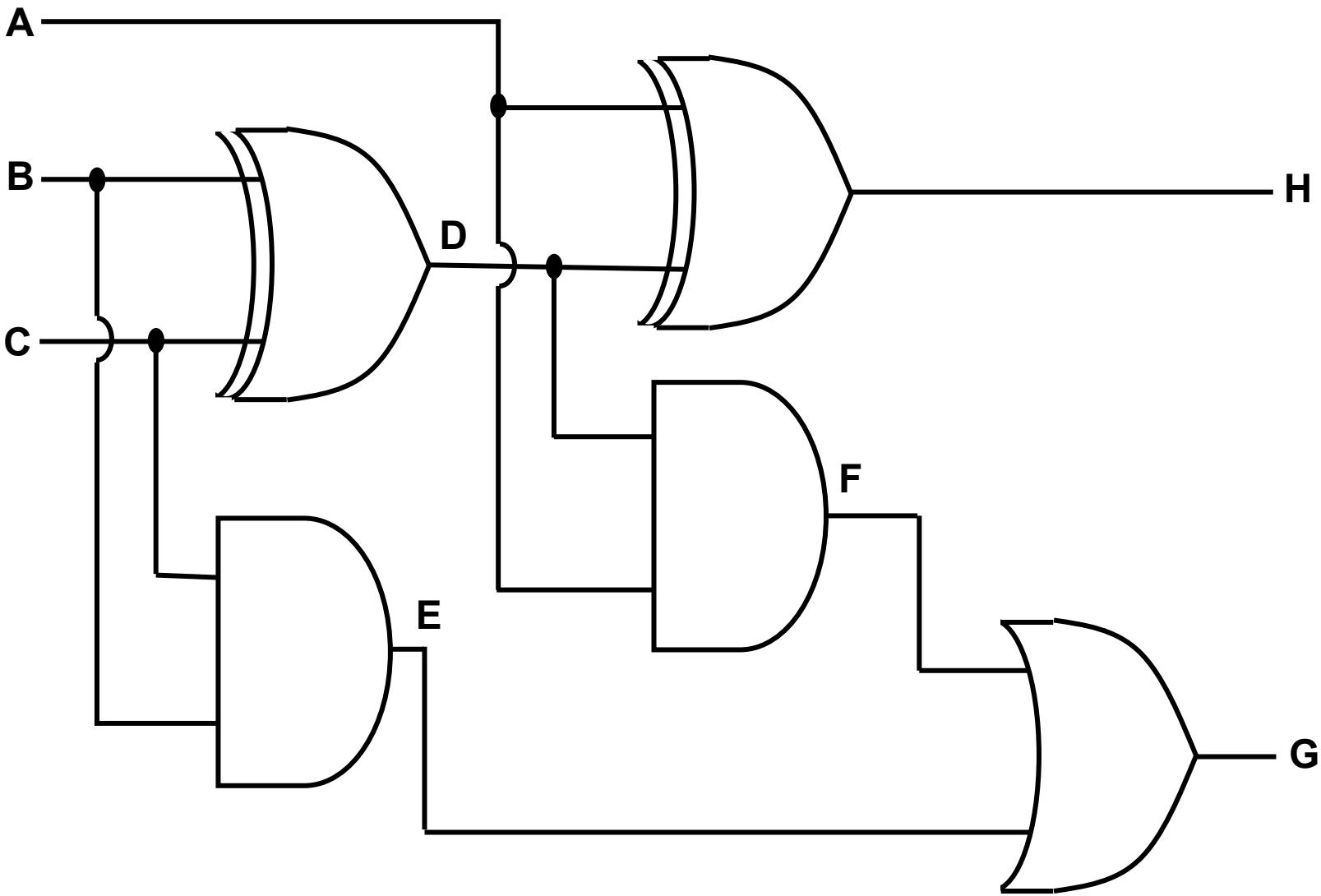
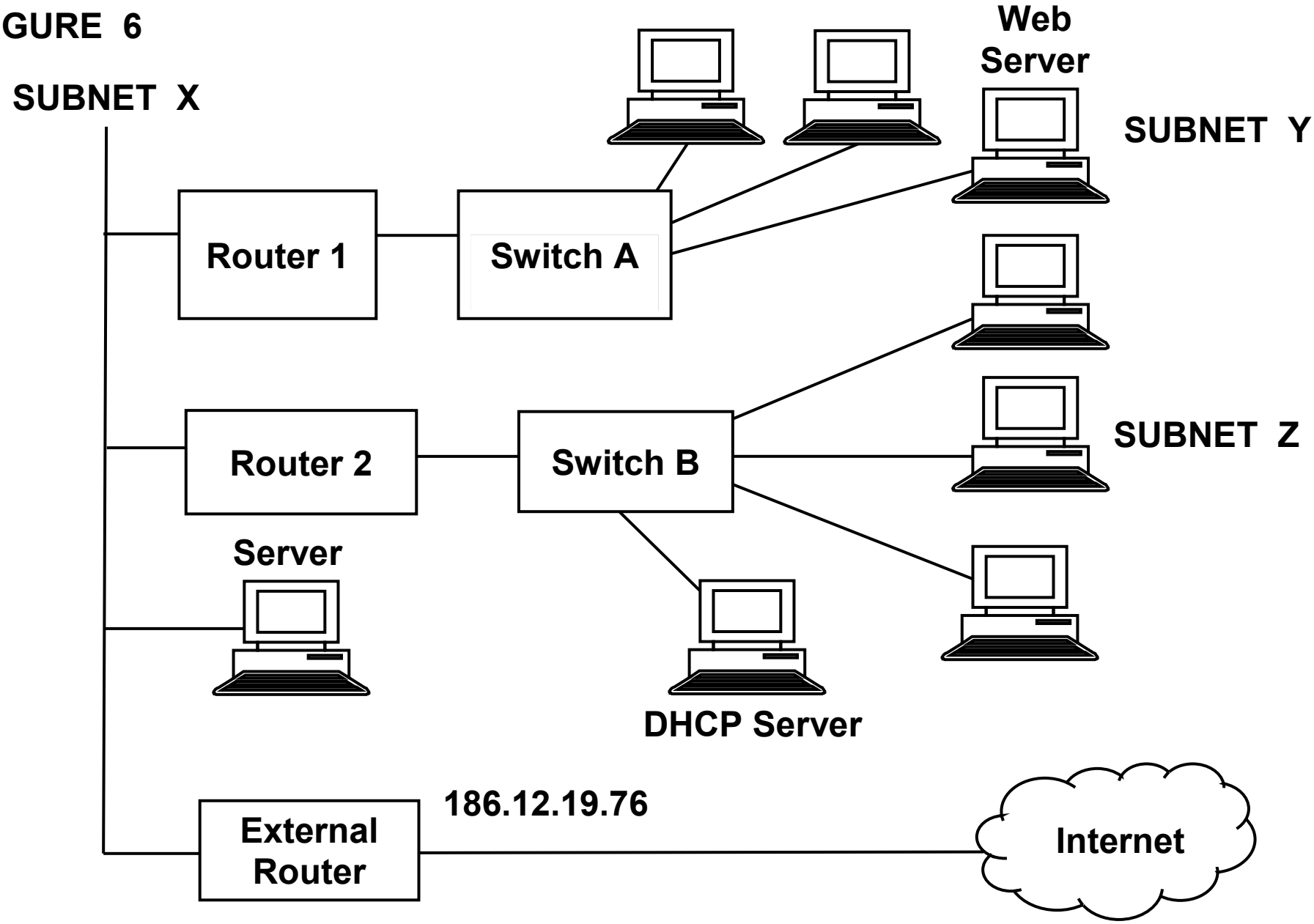


FIGURE 6



[Turn over]

FIGURE 7

Athlete(<u>AthleteID</u>, Surname, Forename, DateOfBirth, Gender, TeamName)

EventType(<u>EventTypeID</u>, Gender, Distance, AgeGroup)

Fixture(<u>FixtureID</u>, FixtureDate, LocationName)

EventAtFixture(<u>FixtureID, EventTypeID</u>)

EventEntry(<u>FixtureID, EventTypeID, AthleteID</u>) 4

- **Each Athlete, EventType and Fixture is identified by a unique identity number, for example AthleteID for athletes.**
- **An EventType is a type of event, such as Boys' 100m Under 15 race.**
- **If an athlete wants to take part in an event at a particular fixture, then an entry is created in the EventEntry relation to represent this.**

FIGURE 11

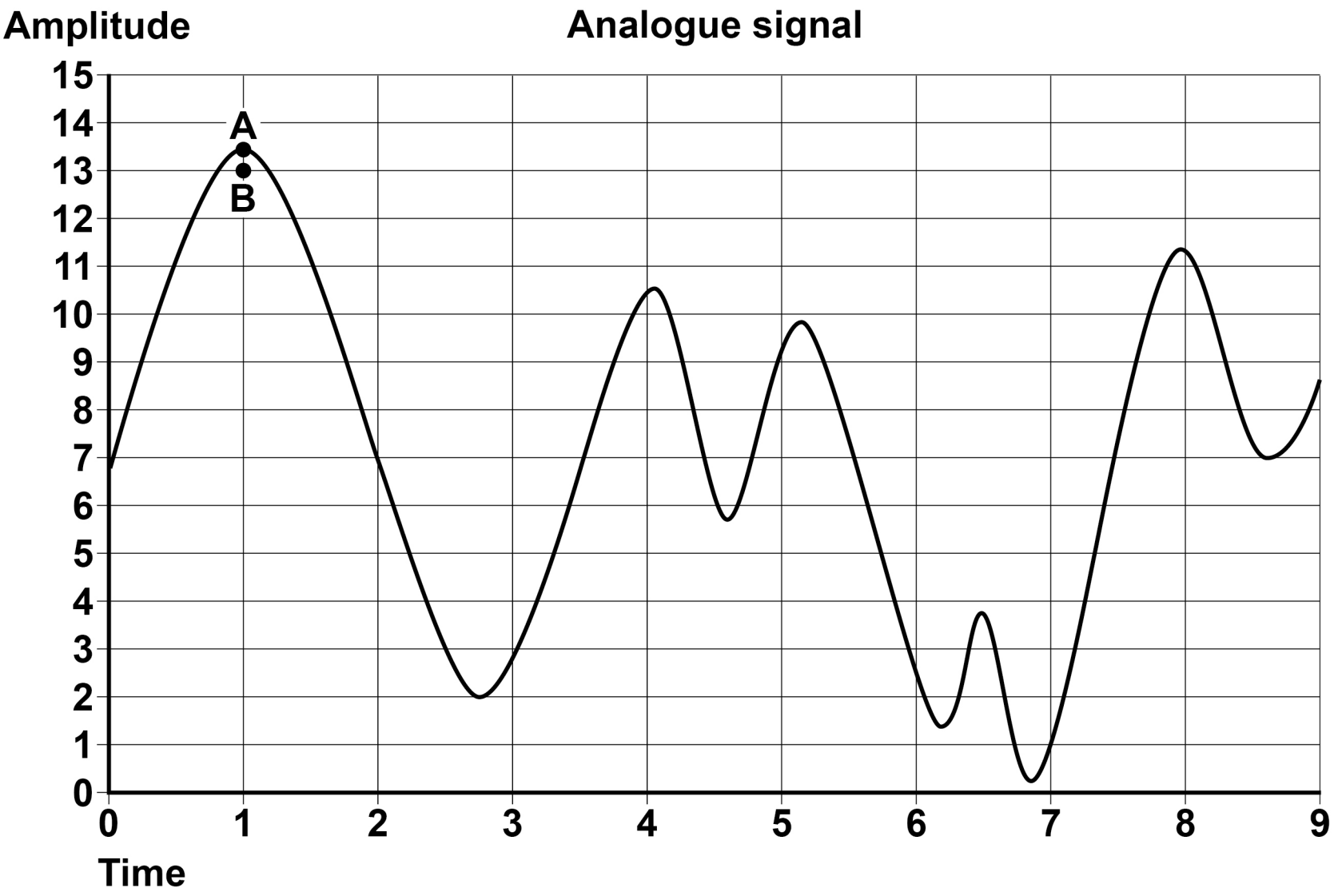

Amplitude
Analogue signal
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
A
B
0
1
2
3
4
5
6
7
8
9
Time

FIGURE 12

```
IF characterCode >= 65 AND characterCode
<= 90 THEN
  encryptedCode ← characterCode +
keyValue
  IF encryptedCode > 90 THEN
    encryptedCode ← encryptedCode - 26
  ENDIF
ELSE
  encryptedCode ← characterCode
ENDIF
```

TABLE 3

A	65
B	66
C	67
D	68
E	69
F	70
G	71
H	72
I	73
J	74
K	75
L	76
M	77

N	78
O	79
P	80
Q	81
R	82
S	83
T	84
U	85
V	86
W	87
X	88
Y	89
Z	90

TABLE 4 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**.**

[Turn over]

`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.**

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 to 12.

END OF SOURCES

There are no sources printed on this page