Ella writes a program on her home computer and compiles it into an executable file.

0 1 . 1 Ella's executable file will not run on Josephine's computer because the two computers have different processors.

Explain why having different processors may have caused this problem.

[2 marks]

The processor in Ella's computer has four cores running at 2.8 GHz and the processor in Josephine's computer has one core running at 3.2 GHz.

0 1 . 2 Considering these differences, explain why Josephine's computer might be able to complete a particular task more quickly than Ella's.

[2 marks]

0 2 **Table 3 – standard AQA assembly language instruction set**. This should be used to answer question part 0 2 . 1

`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.

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.

Figure 3 shows an incomplete assembly language program, intended to perform integer division by 10.

The program decrements the value in R1 in steps of 10 until the value stored in R1 is less than 10. Each time that the value in R1 is decreased by 10 the value in R3 is increased by 1. For example, if R1 started at 43 the sequence of numbers stored in R1 would be 43, 33, 23, 13, 3 and the final value in R3 would be 4.

0 2 . 1 Complete the program in **Figure 3**.

You should assume that R1 has already been assigned a value to divide.

You may not need to use all four lines for your solution and you should not write more than one instruction per line.

[4 marks]

Figure 3

```
                MOV R3, #0

loopstart:      CMP R1, #10

                ____________________

                ____________________

                ____________________

                ____________________

end:            HALT
```

A processor supports 32 different basic machine code operations, and two addressing modes represented by a single bit, as shown in **Figure 4** below.

Figure 4

Opcode						**Operand**									
Basic machine operation					Addressing mode										
0	0	0	0	1	1	0	0	1	1	1	0	1	1	0	1

0 2 . 2 How many different opcodes is the machine potentially capable of supporting?
[1 mark]

0 2 . 3 In direct addressing, the value stored in the operand is the address of the memory location which contains the data to process.

In direct addressing mode, how many memory locations could a processor that used the instruction format described in **Figure 4** potentially make use of?
[1 mark]

0 3 When the processor writes data to the main memory it will make use of the address, control and data buses.

Explain how **each** of these buses will be used during this **write** process.

[4 marks]

Table 1 shows the standard AQA assembly language instruction set that should be used to answer question part 0 4 . 1

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.

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

0 4 **Figure 2** shows an algorithm, written in pseudo-code, that is used to multiply two variables `W` and `X` together. The resulting answer is stored in variable `Y`. It can be assumed that both `W` and `X` are positive integers. `Z` is a temporary variable. The operation DIV performs integer division.

Line numbers are included but are not part of the algorithm.

Figure 2

```
1  W ← 9
2  X ← 12
3  Y ← 0
4  REPEAT
5    Z ← W LOGICAL BITWISE AND 1
6    IF Z = 1 THEN
7      Y ← Y + X
8    END IF
9    W ← W DIV 2
10   X ← X * 2
11 UNTIL W = 0
```

0 4 . 1 Write a sequence of assembly language instructions that perform multiplication using the same method shown in **Figure 2**.

Assume that registers 0, 1, 2 and 3 are used to store the values represented by variables `W`, `X`, `Y` and `Z` accordingly.

Some lines, including those equivalent to line numbers 1 to 5 in **Figure 2**, have been completed for you.

[7 marks

```
            MOV R0, #9
            MOV R1, #12
            MOV R2, #0
startloop:  AND R3, R0, #1




jump:




            B startloop
endloop:
```

0 5 . 1 The memory buffer register and the program counter are examples of registers.

What is a register?

[1 mark]

0 5 . 2 Describe the stored program concept.

[2 marks]

0 5 . 3 Some buses in a computer system have to be bidirectional, meaning data or instructions can travel both ways.

Explain why the data bus in a computer system must be bidirectional.

[2 marks]

0 5 . 4 State **two** differences between how the Harvard and von Neumann architectures operate.

[2 marks]

Difference 1 ____________________

____

____

Difference 2 ____________________

____

____

0 5 . 5 Describe **four** steps that a processor goes through during the fetch stage of the Fetch-Execute cycle.

You **must** explain the purpose of each step.

[8 marks]

Table 1 shows the standard AQA assembly language instruction set that should be used to answer question part 0 6 . 1

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.

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

0 6 . 1 Write an assembly language program to encrypt a single character using the Caesar cipher. The character to be encrypted is represented using a character set consisting of 26 characters with character codes 0–25. The output of the process should be the character code of the encrypted character.

The assembly language instruction set that you should use to write the program is listed in **Table 1**.

Table 2 shows the character codes and the characters they represent.

Table 2

Code	Character	Code	Character	Code	Character
0	A	9	J	18	S
1	B	10	K	19	T
2	C	11	L	20	U
3	D	12	M	21	V
4	E	13	N	22	W
5	F	14	O	23	X
6	G	15	P	24	Y
7	H	16	Q	25	Z
8	I	17	R		

- Memory location 100 contains the character code to be encrypted, which is in the range 0–25
- Memory location 101 contains an integer key to be used for encryption, which is in the range 0–25
- The program should store the character code of the encrypted character in memory location 102

[4 marks]

0 6 . 2 An instruction uses immediate addressing.

What is immediate addressing?

[1 mark]

0 6 . 3 Another method of encryption is the Vernam cipher.

Explain why, under the correct conditions, the Vernam cipher is perfectly secure.

[1 mark]

0 7 . 1 The fetch-execute cycle involves the Current Instruction Register (CIR), Control Unit, Memory Address Register (MAR), Memory Buffer Register (MBR) and Program Counter (PC).

Figure 6 lists four events that can take place during one cycle of the fetch-execute cycle. The events are labelled **A** to **D**.

Some events that take place during the fetch-execute cycle are not listed.

Put these events in the order they would occur in the fetch-execute cycle when an ADD instruction is executed.

Write the numbers 1 to 4 beside each description in **Figure 6** to indicate the order in which the events occur. The number 1 should be used to indicate the event that would happen first.

[3 marks]

Figure 6

	Description	**Order (1 to 4)**
A	The contents of the MBR are copied to the CIR.	
B	The contents of the PC are copied to the MAR.	
C	The Control Unit decodes the contents of the CIR.	
D	The result of the calculation is stored.	

0 7 . 2 Describe the role of main memory in the execution of computer programs.

[2 marks]

0 7 . 3 State the name of the processor component that is responsible for performing mathematical operations such as addition and multiplication.

[1 mark]

0 7 . 4 Explain why increasing the data bus width can lead to improvements in processor performance.

[1 mark]

0 7 . 5 Identify the bus that would need to be changed **and** state the change needed so that the maximum amount of memory addressable by the processor would be doubled.

[2 marks]

Bus to change

Change needed

Table 1 shows the standard AQA assembly language instruction set that should be used to answer question 0 8 . 1 and question 0 8 . 2

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.

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

0 8 . 1 Shade **one** lozenge to show which of the assembly instructions in **Figure 7** uses immediate addressing.

[1 mark]

Figure 7

	Instruction	Immediate Addressing
A	`LDR R3, 42`	⬭
B	`MOV R3, #42`	⬭
C	`STR R3, 101`	⬭
D	`SUB R3, R2, R1`	⬭

0 8 . 2 A computer program is required that will multiply the value stored in X by 2 if it is less than 50 and leave it unchanged if it is 50 or more.

The algorithm for this task can be written in pseudocode as:

```
IF X < 50 THEN
  X ← X * 2
ENDIF
```

Write an assembly language program using the AQA assembly language instruction set shown in **Table 1** to carry out this task.

At the start, the value of X is stored in memory location 101

[4 marks]

0 9 . 1 Explain the role of the status register in a processor **and** describe a circumstance that would result in its contents being updated.

[2 marks]

Table 2 shows the standard AQA assembly language instruction set that should be used to answer question **1 0**

Table 2 – 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.

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

10 **Figure 4** shows an algorithm written in pseudo-code. It is used to calculate the value of the contents of variable A multiplied by the contents of variable B.

Line numbers are included in the pseudo-code but are not part of the algorithm.

Figure 4

```
1      A ← 4
2      B ← 3
3      C ← 0
4      WHILE B > 0
5          C ← C + A
6          B ← B – 1
7      ENDWHILE
```

Write a sequence of assembly language instructions that would perform the same function as the pseudo-code in **Figure 4**.

Registers R1, R2 and R3 are used to hold the values of A, B and C respectively. The assembly language code equivalent to line numbers 1 to 3 in **Figure 4** have been completed for you.

[4 marks]

```
MOV R1, #4
MOV R2, #3
MOV R3, #0
```

1 1 A company is redesigning the processor used in a smartwatch it sells. The redesign will allow the company to increase the clock speed of the processor.

The processor executes all software and controls all hardware on the smartwatch. The smartwatch uses a wide range of sensors to continuously collect data about its wearer and environment. To improve accuracy each sensor takes many readings every second and sends them to the processor for averaging. The smartwatch has different software applications to play music, display images and provide a summary of all the sensor data it has stored.

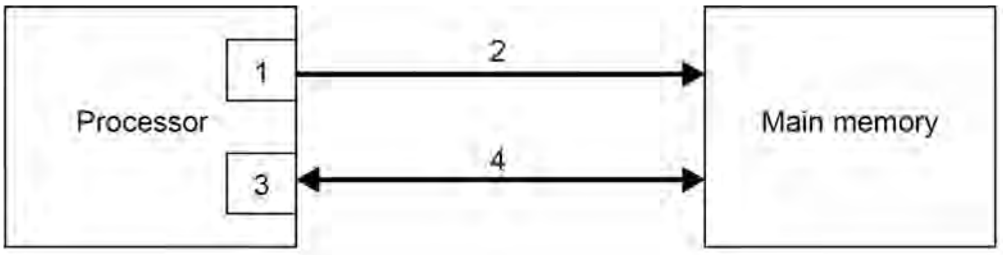
Customer feedback shows that the smartwatch provides all customers with reliable and accurate data. However, some customers mentioned that performance can worsen when loading a large image and listening to music at the same time.

Describe **two** features of the situation that suggest increasing the clock speed would improve the performance of the smartwatch.

[2 marks]

1 2 . 1 **Figure 5** shows some of the processor registers and buses that are used during the fetch stage of the fetch-execute cycle, together with the main memory.

Figure 5



State the name of the components that are labelled in **Figure 5** with the numbers **1** to **4**. In the case of register names, the full names **must** be stated.

[2 marks]

1	
2	
3	
4	

1 2 . 2 Describe the stored program concept.

[2 marks]

1 2 . 3 In a particular processor instruction set, each instruction consists of an opcode and an operand. An operand could be an immediate value to be used by a program.

State **two** other types of value that can be stored in an operand.

[2 marks]

1 2 . 4 Computer A and Computer B both have a processor with a clock speed of 2.8 GHz but Computer A performs tasks much faster than Computer B. Computer A has a larger cache and greater word length than Computer B.

Explain why the larger cache and greater word length are possible factors for the performance difference between Computer A and Computer B.

[2 marks]

Larger cache

Greater word length

Table 1 shows the standard AQA assembly language instruction set that should be used to answer question 1 3

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.

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`:

1. `#` – use the decimal value specified after the #, eg `#25` means use the decimal value 25
2. `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

1 3 Registers R1 and R3 each store a different positive number.

Write a program using the standard AQA assembly language in **Table 1** that will:

- store the greater of these two numbers in R1
- store 1 in R2 if the value originally in R1 is greater than the value in R3, storing 3 in R2 otherwise.

[4 marks]