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

02.2 How many different opcodes is the machine potentially capable of supporting?

[1 mark]

____

02.3 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

1 0 **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.

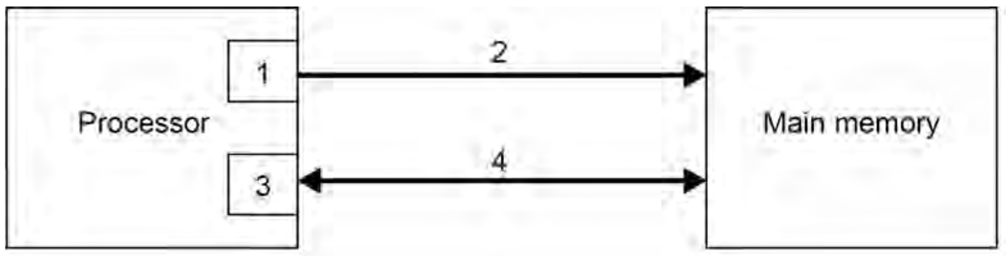
Customer feedback shows that the smartwatch provides all customers with reliable and accurate data. However, some customers mentioned that performance can worsen when loading a large image and listening to music at the same time.

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]

1 4 The greatest common divisor of two positive integers A and B is the largest positive integer that divides both of the numbers without leaving a remainder.

For example, if A = 4 and B = 6 then:

- 4 has the divisors 1, 2 and 4
- 6 has the divisors 1, 2, 3 and 6

Therefore, the greatest common divisor of 4 and 6 is 2, since this is the biggest number which appears in the list of divisors of both 4 and 6.

The method shown in **Figure 7** is a famous method for determining the greatest common divisor of two positive integers, A and B:

Figure 7

```
WHILE A ≠ B
  IF A > B THEN
    A = A – B
  ELSE
    B = B – A
  ENDIF
ENDWHILE
```

When the procedure described in the algorithm terminates, the value in A (and also B) is the greatest common divisor of A and B.

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.

1 4 . 1 Write a program **using the AQA assembly language instruction set**, shown on page 18 in **Table 1**, that uses the method described in **Figure 7** to calculate the greatest common divisor of two positive integers.

- At the start, the positive integer A will be stored in memory location 102 and the positive integer B in memory location 103. Your program should use these values to find their greatest common divisor.
- When your program terminates it should store the greatest common divisor of these two numbers in memory location 104.

[8 marks]

1 5 . 1 A different computer system has a wider data bus; this will speed up the execution of programs.

Explain how the wider data bus has resulted in this effect.

[1 mark]

___

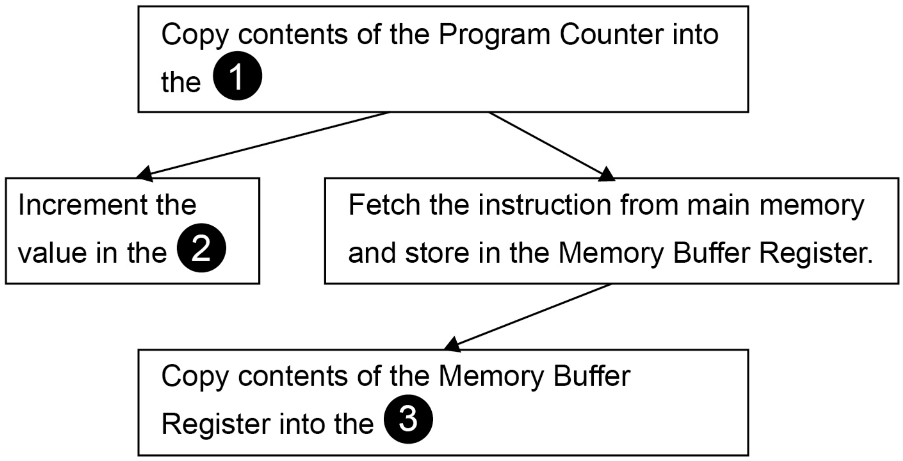
___

___

___

16.1 The diagram in **Figure 2** describes the fetch part of the Fetch-Execute cycle. Some of the names of registers have been omitted from the figure and replaced with the numbers ❶ to ❸

Figure 2


Copy contents of the Program Counter into the 1
Increment the value in the 2
Fetch the instruction from main memory and store in the Memory Buffer Register.
Copy contents of the Memory Buffer Register into the 3

State the **full names** of the registers that should appear in the diagram where the numbers are.

[2 marks]

Number	Full Name of Register
❶	
❷	
❸	

16.2 Interrupts can be generated by devices connected to the processor during the Fetch-Execute cycle.

Describe the role of interrupts.

[2 marks]

1 6 . 3 Explain why the volatile environment (the contents of registers) must be saved before an interrupt is serviced.

[2 marks]

1 7 **Figure 11** shows the format of a machine code instruction for a particular processor and one instruction in that format.

Figure 11

Opcode						**Operand**									
Basic Machine Operation					Addressing Mode										
1	0	0	1	1	1	1	0	0	1	1	1	1	0	1	0

1 7 . 1 If the operand can be used to refer to any location in the memory, how many memory locations can the processor address?

[1 mark]

1 7 . 2 One of the two addressing modes that the processor supports is immediate addressing.

Explain what is meant by immediate addressing.

[1 mark]

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

1 7 . 3 The Vernam cipher encrypts a plaintext character by performing a logical operation between a character in the plaintext and part of the key.

Write an assembly language program, **using the AQA assembly language instruction set** shown on page 28 in **Table 2**, to encrypt a plaintext character using this method.

You should assume that:

- the character code of the plaintext character to be encrypted is stored in memory location 101
- the part of the key to use to encrypt the character is stored in memory location 102

The encrypted ciphertext character should be stored in memory location 103

[3 marks]

This table is included so that you can answer Question 18.1 on page 17.

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

1 8 **Figure 4** shows an assembly language program which has been written using the AQA assembly language instruction set. The instruction set is explained in **Table 1** on page 15.

Figure 4

```
  CMP R2, #0
  BEQ exit
  MOV R0, #0
  MOV R3, #1
moveleft:
  LSL R2, R2, #1
  LSL R3, R3, #1
  CMP R2, R1
  BLT moveleft
  BEQ mainloop
  LSR R2, R2, #1
  LSR R3, R3, #1
mainloop:
  CMP R1, R2
  BLT skip
  ADD R0, R0, R3
  SUB R1, R1, R2
skip:
  AND R4, R3, #1
  CMP R4, #1
  BEQ skipshiftR2
  LSR R2, R2, #1
skipshiftR2:
  LSR R3, R3, #1
  CMP R3, #0
  BNE mainloop
exit:
  HALT
```

The program takes its input values from registers `R1` and `R2` and stores its output in registers `R0` and `R1`

1 8 . 1 Complete the trace table below to show the results of executing the program in **Figure 4** when the initial values in registers R1 and R2 are 34 and 6

Each register can hold a 16-bit value.

You may find it easier to understand the operation of the program if you write the contents of the registers out in both binary and decimal.

You may not need to use all the rows in the table.

[6 marks]

R0	R1	R2	R3	R4
	100010 (34)	110 (6)		

18.2 The initial values for the program (its inputs) are stored in `R1` and `R2` and the final values stored in `R0` and `R1` are its outputs.

By considering the inputs and the outputs in your trace table for Question **06.1**, describe the purpose of the program.

[2 marks]

1 9 . 1 The control unit is an important component of a processor.

Describe the role of the control unit.

[3 marks]

1 9 . 2 One method that can be used to improve the performance of a processor is to increase the amount of cache memory.

Describe:

- what cache memory is
- what cache memory is used for
- how increasing the amount of cache memory can improve the performance of a processor.

[4 marks]

This table is included so that you can answer Questions 20.1 and 20.2 on page 21.

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

2 0 **Figure 7** shows an assembly language program that has been written using the AQA Assembly Language Instruction Set, which is given in **Table 1** on **page 20**.

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
```

2 0 . 1 State the name of the addressing mode used in the instruction `ADD R3, R3, R0`

[1 mark]

2 0 . 2 Memory location 120 contains the value 23 and memory location 121 contains the value 5.

Complete the trace table to show how the contents of the memory locations and registers change when the program in **Figure 7** is executed.

[5 marks]

Memory locations			**Registers**			
120	121	122	R0	R1	R2	R3
23	5					

2 0 . 3 State the purpose of the program in **Figure 7**.

[1 mark]

2 0 . 4 The program in **Figure 7** has been written using assembly language.

State **two** reasons why the programmer may have chosen to write this program in assembly language rather than in a high-level programming language.

[2 marks]

Reason 1

Reason 2

2 0 . 5 The program in **Figure 7** will be translated into machine code.

Explain the relationship between an assembly language instruction and a machine code instruction.

[1 mark]

2 1 . 1 Describe how the fetch-execute cycle is used to carry out machine code instructions **and** how the hardware of a computer could be improved so that programs can be executed more quickly.

Your response should include a description of what happens during each stage of the fetch-execute cycle.

[12 marks]

2 1 . 2 An interrupt may occur during the fetch-execute cycle.

Describe what an interrupt is **and** explain the purpose of interrupts.

[2 marks]

2 2

For question parts **22.1** and **22.2** you should assume that memory locations and registers store **8-bit** values. These question parts use the AQA assembly language instruction set in **Table 3** on **page 23**.

Assembly language instructions can be used to perform masking, which allows the values of individual bits or groups of bits within a number to be isolated or set independently of the values of the other bits in the number.

For example, to isolate the values of the rightmost four bits of an 8-bit number, the number could be ANDed with the binary value `00001111`.

The assembly language instruction `AND R0, R1, #15` performs a bitwise logical AND operation between the value in register `R1` and the number `15` (equivalent to `00001111` in binary), storing the result in register `R0`.

2 2 . 1 In binary, show the result of applying the instruction `AND R0, R1, #15` when register `R1` contains the decimal value 70 which is 46 in hexadecimal.

[1 mark]

R1	0	1	0	0	0	1	1	0
15	0	0	0	0	1	1	1	1
R0								

2 2 . 2 In binary, show the result of applying the instruction `ORR R0, R1, #48` when register `R1` contains the decimal value 6 which is 6 in hexadecimal.

[1 mark]

R1	0	0	0	0	0	1	1	0
48	0	0	1	1	0	0	0	0
R0								

2 2 . 3 A computer program is required to display the value of the contents of a memory location that stores an 8-bit value. The value should be displayed on the screen of the computer in hexadecimal.

Part of the process required to do this is to convert the value stored in the memory location into the correct ASCII codes for each of the two digits that represent that value in hexadecimal.

For example, if the memory location contained:

1	0	0	1	1	1	1	0

which is 9E in hexadecimal, then the ASCII codes of the characters that need to be displayed are:

0	0	1	1	1	0	0	1

0	1	0	0	0	1	0	1

The first of these is the ASCII code of the character 9, the second is the ASCII code of the character E.

Write an assembly language program using the AQA assembly language instruction set that will load a value from memory location 100 and store the ASCII code of the first (lefthand) digit of the hexadecimal representation of this value in memory location 101 and the ASCII code of the second (righthand) digit of the hexadecimal representation of this value in memory location 102.

Your program should use masking and/or shifting to complete this task.

The ASCII codes of the hexadecimal digits are shown in **Table 2** and the AQA assembly language instruction set is in **Table 3** on **page 23**.

Table 2

	ASCII Code	
Digit	**Decimal**	**Binary**
0	48	0110000
1	49	0110001
2	50	0110010
3	51	0110011
4	52	0110100
5	53	0110101
6	54	0110110
7	55	0110111

	ASCII Code	
Digit	**Decimal**	**Binary**
8	56	0111000
9	57	0111001
A	65	1000001
B	66	1000010
C	67	1000011
D	68	1000100
E	69	1000101
F	70	1000110

[10 marks]

This table is included so that you can answer question parts 22.1, 22.2 and 22.3. Table 3 – 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

This table is included so that you can answer Question 23.1 on page 35.

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

2 3 **Figure 7** shows an assembly language program which has been written using the AQA assembly language instruction set. The instruction set is explained in **Table 1** on page 33.

Figure 7

```
  LDR R1, 130
  MOV R2, #0
  MOV R4, #0
repeat:
  ADD R2, R2, #1
  AND R3, R1, #1
  CMP R3, #0
  BEQ skip
  ADD R4, R4, #1
skip:
  LSR R1, R1, #1
  CMP R2, #7
  BNE repeat
  LDR R1, 130
  AND R4, R4, #1
  CMP R4, #0
  BNE else
  ORR R1, R1, #128
  B end
else:
  AND R1, R1, #127
end:
  STR R1, 130
  HALT
```

The program performs a task on a value stored in memory location 130. The value in this memory location is a 7-bit ASCII code.

For example, if memory location 130 was used to store the ASCII character ‘S’ then it would contain the value 83, which in binary is:

0	1	0	1	0	0	1	1

2 3 . 1 Complete the trace table below to show the results of executing the program in **Figure 7** when the initial value in memory location `130` is 83

Each register can hold an 8-bit value.

You may find it easier to understand the operation of the program if you write the contents of memory location `130` and register `R1` out in both binary and decimal.

You may not need to use all the rows in the table.

[6 marks]

Memory Location 130	**R1**	**R2**	**R3**	**R4**
83 (01010011)				

2 3 . 2 The value in memory location `130` before the program is executed is the program's input and the value stored in memory location `130` when the program finishes executing is its output.

By considering your trace table for Question **10.1** and the assembly language code in **Figure 7**, describe the purpose of the program.

[2 marks]