QUESTION 1.

5 A gardener grows vegetables in a greenhouse. For the vegetables to grow well, needs to always be within a particular range.

The gardener is not sure about the actual temperatures in the greenhouse during the season. The gardener installs some equipment. This records the temperature every hour o the growing season.

(a) Name the type of system described.

....[1]

(b) Identify **three** items of hardware that would be needed to acquire and record the temperature data. Justify your choice for each.

Item 1 ....

Justification ....

....

Item 2 ....

Justification ....

....

Item 3 ....

Justification ....

....[6]

(c) The equipment records temperatures in the greenhouse. It does this for seven locations.

Each recording is stored as two successive bytes. The format is shown below:

Greenhouse location

7	6	5	4	3	2	1	0

Byte 1

Temperature reading

Byte 2

The location is indicated by the setting of one of the seven bits in byte 1. For example, location 4 is indicated by setting bit 4.

Bit 0 of byte 1 acts as a flag:

- the initial value is zero
- when the reading has been processed it is set to 1

Byte 2 contains the temperature reading (two's complement integer).

(i) Interpret the data in byte 1 shown below:

7	6	5	4	3	2	1	0
0	0	1	0	0	0	0	1

Byte 1

0	0	0	1	1	0	0	0

Byte 2

....

....

....

....[2]

(ii) The system receives a temperature reading of –5 degrees from sensor 6.

Complete the boxes below to show the two bytes for this recording. The reading has not yet been processed.

7	6	5	4	3	2	1	0

Byte 1

Byte 2

[2]

(d) **(i)** The accumulator is loaded with the value of byte 1 from location 106.

Write the assembly language instruction to check whether the reading in byte 2 came from location 4.

```
LDD 106             // data loaded from address 106
```

....[4]

(ii) Write the assembly language instruction to set the flag (bit 0) of the byte contained in the accumulator to 1.

....[2]

QUESTION 2.

6 A company grows vegetables in a number of large greenhouses. For the vegetab the temperature, light level and soil moisture need to always be within certain range

The company installs a computerised system to keep these three growing conditions w best ranges. Sensors are used for collecting data about the temperature, light level, and mo content of the soil.

(a) Name the type of system described.

....[1]

(b) Give **three** items of hardware that would be needed for this system. Justify your choice. Do not include sensors in your answer.

Item 1 ....

Justification ....

....

Item 2 ....

Justification ....

....

Item 3 ....

Justification ....

....[6]

(c) **(i)** Describe what is meant by feedback in the above system.

....

....

....

....

....

....[3]

(ii) When the system was designed, various parameters for temperature were set.

Name **one** of these parameters.

....

....[1]

(iii) Explain how this parameter value is used by the feedback system.

....

....

....

....[2]

Question 6 continues on page 14.

Each greenhouse has eight sensors (numbered 1–8).

- The byte at address 150 is used to store eight 1-bit flags.
- A flag is set to indicate whether its associated sensor reading is waiting to be processe
- More than one sensor reading may be waiting to be processed at any particular moment.
- Data received from the sensors is stored in a block of eight consecutive bytes (addresses 201–208).
- The data from sensor 1 is at address 201, the data from sensor 2 is at address 202, and so on.

Sensor number

	1	2	3	4	5	6	7	8
`150`	0	1	0	0	0	1	0	1

`201`	0	0	0	0	0	0	0	0
`202`	0	0	0	0	0	1	0	0
`203`	0	0	0	0	0	0	0	0
`204`	0	0	0	1	0	0	0	0
`205`	0	0	0	0	0	0	1	0
`206`	0	0	0	1	0	1	0	0
`207`	0	0	0	1	0	0	1	0
`208`	0	0	0	1	0	0	1	0

(d) **(i)** Interpret the current reading for sensor 2.

....

....[2]

(ii) The accumulator is loaded with the data from location 150.

Write the assembly language instruction to check whether there is a value waiting to be processed for sensor 6.

```
LDD 150                  // data loaded from address 150
```

....[3]

BLANK PAGE

QUESTION 3.

2 A compiler uses a keyword table and a symbol table. Part of the keyword table is

- Tokens for keywords are shown in hexadecimal.
- All the keyword tokens are in the range 00 – 5F.

Keyword	Token
←	01
+	02
=	03
IF	4A
THEN	4B
ENDIF	4C
ELSE	4D
FOR	4E
STEP	4F
TO	50
INPUT	51
OUTPUT	52
ENDFOR	53

Entries in the symbol table are allocated tokens. These values start from 60 (hexadecimal).

Study the following piece of code:

```
Counter ← 1.5
INPUT Num1
  // Check values
IF Counter = Num1
   THEN
      Num1 ← Num1 + 5.0
ENDIF
```

(a) Complete the symbol table below to show its contents after the lexical analysis stage.

Symbol	Token	
	Value	Type
Counter	60	Variable
1.5	61	Constant

[3]

(b) Each cell below represents one byte of the output from the lexical analysis s

Using the keyword table and your answer to **part (a)** complete the output from analysis.

60	01														

[2]

(c) This line of code is to be compiled:

```
A ← B + C + D
```

After the syntax analysis stage, the compiler generates object code. The equivalent code, in assembly language, is shown below:

```
LDD 234     //loads value B
ADD 235     //adds value C
STO 567     //stores result in temporary location
LDD 567     //loads value from temporary location
ADD 236     //adds value D
STO 233     //stores result in A
```

(i) Name the final stage in the compilation process that follows this code generation stage.

....[1]

(ii) Rewrite the equivalent code given above to show the effect of it being processed through this final stage.

....

....

....

....

....

....[2]

(iii) State **two** benefits of the compilation process performing this final stage.

Benefit 1 ....

....

Benefit 2 ....

....[2]

QUESTION 4.

2 In this question, you are shown pseudocode in place of a real high-level langu uses a keyword table and a symbol table. Part of the keyword table is shown below.

- Tokens for keywords are shown in hexadecimal.
- All the keyword tokens are in the range 00 to 5F.

Keyword	Token
←	01
+	02
=	03
IF	4A
THEN	4B
ENDIF	4C
ELSE	4D
FOR	4E
STEP	4F
TO	50
INPUT	51
OUTPUT	52
ENDFOR	53

Entries in the symbol table are allocated tokens. These values start from 60 (hexadecimal).

Study the following piece of code:

```
Start ← 0.1
// Output values in loop
FOR Counter ← Start TO 10
   OUTPUT Counter + Start
ENDFOR
```

(a) Complete the symbol table below to show its contents after the lexical analysis stage.

Symbol	Token	
	Value	**Type**
Start	60	Variable
0.1	61	Constant

[3]

(b) Each cell below represents one byte of the output from the lexical analysis s

Using the keyword table and your answer to **part (a)** complete the output from analysis.

60	01												

[2]

(c) The compilation process has a number of stages. The output of the lexical analysis stage forms the input to the next stage.

(i) Name this stage.

....[1]

(ii) State **two** tasks that occur at this stage.

....

....

....

....[2]

(d) The final stage of compilation is optimisation. There are a number of reasons for performing optimisation. One reason is to produce code that minimises the amount of memory used.

(i) State another reason for the optimisation of code.

....[1]

(ii) What could a compiler do to optimise the following expression?

```
A ← B + 2 * 6
```

....

....

....[1]

(iii) These lines of code are to be compiled:

```
X ← A + B
Y ← A + B + C
```

Following the syntax analysis stage, object code is generated. The equivalent code, assembly language, is shown below:

```
LDD 436    //loads value A
ADD 437    //adds value B
STO 612    //stores result in X
LDD 436    //loads value A
ADD 437    //adds value B
ADD 438    //adds value C
STO 613    //stores result in Y
```

(iv) Rewrite the equivalent code, given above, following optimisation.

....

....

....

....

....

....[3]

QUESTION 5.

6 An intruder detection system for a large house has four sensors. An 8-bit memor the output from each sensor in its own bit position.

The bit value for each sensor shows:

- 1 – the sensor has been triggered
- 0 – the sensor has not been triggered

The bit positions are used as follows:

Not used				Sensor 4	Sensor 3	Sensor 2	Sensor 1

The output from the intruder detection system is a loud alarm.

(a) **(i)** State the name of the type of system to which intruder detection systems belong.

....[1]

(ii) Justify your answer to **part (i)**.

....

....[1]

(b) Name **two** sensors that could be used in this intruder detection system. Give a reason for your choice.

Sensor 1 ....

Reason ....

....

Sensor 2 ....

Reason ....

....[4]

The intruder system is set up so that the alarm will only sound if two or more se triggered.
An assembly language program has been written to process the contents of the memo

The table shows part of the instruction set for the processor used.

Instruction		Explanation
Op code	Operand	
`LDD`	`<address>`	Direct addressing. Load the contents of the given address to ACC
`STO`	`<address>`	Store the contents of ACC at the given address
`INC`	`<register>`	Add 1 to the contents of the register (ACC or IX)
`ADD`	`<address>`	Add the contents of the given address to the contents of ACC
`AND`	`<address>`	Bitwise AND operation of the contents of ACC with the contents of <address>
`CMP`	`#n`	Compare the contents of ACC with the number n
`JMP`	`<address>`	Jump to the given address
`JPE`	`<address>`	Following a compare instruction, jump to <address> if the compare was True
`JGT`	`<address>`	Following a compare instruction, jump to <address> if the content of ACC is greater than the number used in the compare instruction
`END`		End the program and return to the operating system

(c) Part of the assembly code is:

	Op code	Operand
SENSORS:		B00001010
COUNT:		0
VALUE:		1
LOOP:	LDD	SENSORS
	AND	VALUE
	CMP	#0
	JPE	ZERO
	LDD	COUNT
	INC	ACC
	STO	COUNT
ZERO:	LDD	VALUE
	CMP	#8
	JPE	EXIT
	ADD	VALUE
	STO	VALUE
	JMP	LOOP
EXIT:	LDD	COUNT
TEST:	CMP	…
	JGT	ALARM

(i) Dry run the assembly language code. Start at `LOOP` and finish when `EXI`

BITREG	**COUNT**	**VALUE**	**ACC**
B00001010	0	1	

[4]

(ii) The operand for the instruction labelled `TEST` is missing.

State the missing operand.

....[1]

(iii) The intruder detection system is improved and now has eight sensors.

One instruction in the assembly language code will need to be amended.

Identify this instruction ....

Write the amended instruction ....[2]

QUESTION 6.

6 A large warehouse stores goods that must be kept above a temperature of 15 degrees Celsius. The warehouse has six temperature sensors which are each placed at a different location in the warehouse.

A computer system is programmed to turn on appropriate heaters when one of the sensors is below the minimum temperature.

(a) **(i)** State the name given to the type of system described.

.... [1]

(ii) Justify your answer to **part (i)**.

....

.... [1]

(b) Sensors and heaters are two types of device used in this system.

State **two** other devices that are used. Justify your choice.

Device 1 ....

Justification ....

....

Device 2 ....

Justification ....

....

[4]

(c) The computer system stores the temperature readings for the six sensors in locations.

Six of the bits in an 8-bit register, `LOWREG`, are used to indicate whether a particular is below the minimum temperature. A value of `1` means the reading is below the min temperature.

For example:

This pattern of bits in `LOWREG` shows that sensor 5, sensor 4 and sensor 1 have readings below the minimum temperature.

		6	5	4	3	2	1
Not used	Not used	0	1	1	0	0	1

The following table shows part of the instruction set for a processor which has one general purpose register, the Accumulator (ACC), and an Index Register (IX).

Instruction		**Explanation**
Op code	**Operand**	
`LDD`	`<address>`	Direct addressing. Load the contents of the given address to ACC.
`LDR`	`#n`	Immediate addressing. Load the number n to IX.
`LDX`	`<address>`	Indexed addressing. Form the address from <address> + the contents of the index register. Copy the contents of this calculated address to ACC.
`STO`	`<address>`	Store the contents of ACC at the given address.
`INC`	`<register>`	Add 1 to the contents of the register (ACC or IX).
`ADD`	`<address>`	Add the contents of the given address to the ACC.
`OR`	`<address>`	Bitwise OR operation of the contents of ACC with the contents of address.
`CMP`	`#n`	Compare the contents of ACC with number n.
`CMP`	`<address>`	Compare the contents of ACC with the contents of <address>.
`JMP`	`<address>`	Jump to the given address.
`JPE`	`<address>`	Following a compare instruction, jump to <address> if the compare was True.
`JGE`	`<address>`	Following a compare instruction, jump to <address> if the content of ACC is greater than or equal to the number used in the compare instruction.

Question 6(c) continues on the next page.

Part of the assembly language code for updating LOWREG is:

Label	Op code	Operand
LOWTEMP:		15
LOWREG:		B00000000
COUNTER:		1
START:	LDR	#0
LOOP:	LDX	8000
	CMP	LOWTEMP
	JGE	TEMPOK
	LDD	LOWREG
	OR	COUNTER
	STO	LOWREG
TEMPOK:	LDD	COUNTER
Q1:	CMP	#32
	JPE	HEATON
	ADD	COUNTER
	STO	COUNTER
	INC	IX
	JMP	LOOP
HEATON:	LDD	LOWREG

(i) The code uses six memory locations to store the temperature readings. I
for sensors 1 to 6 at addresses 8000 to 8005.

At a particular time, the memory locations store the following data.

8000	8001	8002	8003	8004	8005
17	14	15	15	16	14

Dry run the assembly language code starting at START and finishing when the loop has been processed twice.

LOWTEMP	**LOWREG**	**COUNTER**	**ACC**	**IX**
15	B00000000	1		

[4]

QUESTION 7.

6 The environment in a very large greenhouse is managed by a computer system. a number of different sensors that include temperature sensors. In addition, the syst number of heaters, windows and sprinklers.

(a) State **one** other type of sensor that could be used with this system.

Justify your choice.

Sensor ....

Justification ....

....

[2]

(b) Describe why feedback is important in this system.

....

....

....

....

....

....[3]

(c) **(i)** The system makes use of a number of parameters. These parameters are used in the code that runs the system.

State **one** of the parameters used in controlling the temperature in the greenhouse.

....[1]

(ii) Explain how the parameter identified in **part (c)(i)** is used in the feedback process.

....

....

....

....[2]

(d) There are eight temperature sensors numbered 1 to 8. Readings from th stored in four 16-bit memory locations. The memory locations have addresses 4003. Each memory location stores two sensor readings as two unsigned binary i

Sensor 1 reading is stored in bits 8 to 15 of address 4000; Sensor 2 reading is stored i 0 to 7 of address 4000 and so on. The diagram shows that the current sensor 1 reading i a value of 97.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
4000	0	1	1	0	0	0	0	1	0	0	1	1	1	0	0	1
4001	1	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0
4002	0	0	0	1	0	1	0	0	0	0	0	0	1	1	0	1
4003	1	0	0	0	0	0	1	0	1	1	0	0	0	1	0	1

(i) Give the denary value of the current reading for Sensor 5.

....

....

....

....[1]

(ii) The following table shows part of the instruction set for a processor. Th
one general purpose register, the Accumulator (ACC).

Instruction		Explanation
Op code	Operand	
`LDD`	`<address>`	Direct addressing. Load the contents of the location at the given address to ACC.
`AND`	`#n`	Bitwise AND operation of the contents of ACC with the operand.
`AND`	`<address>`	Bitwise AND operation of the contents of ACC with the contents of <address>.
`XOR`	`#n`	Bitwise XOR operation of the contents of ACC with the operand.
`XOR`	`<address>`	Bitwise XOR operation of the contents of ACC with the contents of <address>.
`OR`	`#n`	Bitwise OR operation of the contents of ACC with the operand.
`OR`	`<address>`	Bitwise OR operation of the contents of ACC with the contents of <address>. <address> can be an absolute address or a symbolic address.
`LSL`	`#n`	Bits in ACC are shifted n places to the left. Zeros are introduced on the right hand end.
`LSR`	`#n`	Bits in ACC are shifted n places to the right. Zeros are introduced on the left hand end.

The reading for Sensor 5 is used in a calculation. The calculation is carried out by two assembly language instructions.

The first instruction loads the contents of the 16-bit location that contains the value for Sensor 5.

The second instruction moves the bits in Sensor 5 so that the 16-bit value is the value of Sensor 5.

Complete the two instructions in the following code. Use the instruction set provided.

```
LDD ........................... // load the contents of the 16-bit location
                                   containing the value for Sensor 5 into the
                                   Accumulator

...................................// move the bits in the Accumulator so that the
                                   Accumulator stores the value of Sensor 5 as an
                                   unsigned 16-bit binary integer
```

[3]

QUESTION 8.

6 The compilation process has a number of stages. The first stage is lexical analys

A compiler uses a keyword table and a symbol table. Part of the keyword table is show

- Tokens for keywords are shown in hexadecimal.
- All of the keyword tokens are in the range `00` – `5F`.

Keyword	Token
←	01
*	02
=	03
IF	4A
THEN	4B
ENDIF	4C
ELSE	4D
FOR	4E
STEP	4F
TO	50
INPUT	51
OUTPUT	52
ENDFOR	53

Entries in the symbol table are allocated tokens. These values start from 60 (hexadecimal).

Study the following code.

```
Start ← 1
INPUT Number
// Output values in a loop
FOR Counter ← Start TO 12
    OUTPUT Number * Counter
ENDFOR
```

(a) Complete the symbol table to show its contents after the lexical analysis stag

Symbol	Token	
	Value	Type
`Start`	`60`	Variable
`1`	`61`	Constant

[3]

(b) The output from the lexical analysis stage is stored in the following table. Each cell stores one byte of the output.

Complete the output from the lexical analysis stage. Use the keyword table and your answer to **part (a)**.

`60`	`01`														

[2]

(c) The output of the lexical analysis stage is the input to the syntax analysis stage.

Identify **two** tasks in syntax analysis.

1 ....

....

2 ....

....
[2]

(d) The final stage of compilation is optimisation.

(i) Code optimisation produces code that minimises the amount of memory used.

Give **one** additional reason why code optimisation is performed.

....

....[1]

(ii) A student uses the compiler to compile some different code.

After the syntax analysis stage is complete, the compiler generates object co

The following lines of code are compiled.

```
X  ←  A + B
Y  ←  A + B + C
Z  ←  A + B + C + D
```

The compilation produces the following assembly language code.

```
LDD 236     //     loads value A to accumulator
ADD 237     //     adds value B to accumulator
STO 512     //     stores accumulator in X
LDD 236     //     loads value A to accumulator
ADD 237     //     adds value B to accumulator
ADD 238     //     adds value C to accumulator
STO 513     //     stores accumulator in Y
LDD 236     //     loads value A to accumulator
ADD 237     //     adds value B to accumulator
ADD 238     //     adds value C to accumulator
ADD 239     //     adds value D to accumulator
STO 514     //     stores accumulator in Z
```

Rewrite the assembly language code after it has been optimised.

....

....

....

....

....

....

....

....

....

.... [5]

BLANK PAGE

QUESTION 9.

4 A compiler uses a keyword table and a symbol table. Part of the keyword table is

- Tokens for keywords are shown in hexadecimal.
- All of the keyword tokens are in the range `00 – 5F`.

Keyword	Token
←	01
+	02
=	03
IF	4A
THEN	4B
ENDIF	4C
ELSE	4D
FOR	4E
STEP	4F
TO	50
INPUT	51
OUTPUT	52
ENDFOR	53

Entries in the symbol table are allocated tokens. These values start from `60` (hexadecimal).

Study the following code.

```
INPUT Number1
INPUT Number2
INPUT Answer
IF Answer = Number1 + Number2
   THEN
      OUTPUT 10
   ELSE
      OUTPUT 0
ENDIF
```

(a) Complete the symbol table to show its contents after the lexical analysis stage.

Symbol	Token	
	Value	Type
Number1	60	Variable
Number2	61	Variable

[3]

(b) The output from the lexical analysis stage is stored in the following table. Eac byte of the output.

Complete the output from the lexical analysis. Use the keyword table and your a **part (a)**.

51	60																	

[2]

(c) A student uses the compiler to compile some different code.

After the syntax analysis is complete, the compiler generates object code.

The following line of code is compiled: `X ← A + B + C − D`

The compilation produces the following assembly language code.

```
LDD 236      // loads value A into accumulator
ADD 237      // adds value B to accumulator
ADD 238      // adds value C to accumulator
STO 540      // stores accumulator in temporary location
LDD 540      // loads value from temporary location into accumulator
SUB 239      // subtracts value D from accumulator
STO 235      // stores accumulator in X
```

(i) Identify the final stage in the compilation process that follows this code generation stage.

....[1]

(ii) Rewrite the equivalent code following the final stage.

....

....

....

....

....

....

....

....[3]

(iii) State **two** benefits of the process that is carried out in the final stage.

Benefit 1 ....

....

Benefit 2 ....

....

[2]

(d) An interpreter is executing a program. The program uses the variables `a`, `b`, `c` and `d`.

The program contains an expression that is written in infix form. The interpreter converts the infix expression to RPN.

The RPN expression is: `b a c + * d + 2 −`

The interpreter evaluates this RPN expression using a stack.

The current values are: `a = 1` `b = 2` `c = 2` `d = 3`

Show the changing contents of the stack as the interpreter evaluates the expression.

The first entry on the stack has been done for you.

QUESTION 10.

4 A compiler uses a keyword table and a symbol table. Part of the keyword table is

- Tokens for keywords are shown in hexadecimal.
- All of the keyword tokens are in the range 00 – 5F.

Keyword	Token
←	01
+	02
=	03
<>	04
IF	4A
THEN	4B
ENDIF	4C
ELSE	4D
REPEAT	4E
UNTIL	4F
TO	50
INPUT	51
OUTPUT	52
ENDFOR	53

Entries in the symbol table are allocated tokens. These values start from 60 (hexadecimal).

Study the following piece of pseudocode.

```
Counter ← 0
INPUT Password
REPEAT
   IF Password <> "Cambridge"
      THEN
         INPUT Password
   ENDIF
   Counter ← Counter + 1
UNTIL Password = "Cambridge"
OUTPUT Counter
```

(a) Complete the symbol table to show its contents after the lexical analysis stag

Symbol	Token	
	Value	**Type**
`Counter`	`60`	Variable

[3]

(b) The output from the lexical analysis stage is stored in the following table. Each cell stores one byte of the output.

Complete the output from the lexical analysis using the keyword table **and** your answer to **part (a)**.

60	01																							

[2]

(c) The following table shows assembly language instructions for a processo
general purpose register, the Accumulator (ACC).

Instruction		Explanation
Op code	**Operand**	
`LDD`	`<address>`	Direct addressing. Load the contents of the location at the given address to ACC
`ADD`	`<address>`	Add the contents of the given address to the ACC
`STO`	`<address>`	Store the contents of ACC at the given address

After the syntax analysis is completed successfully, the compiler generates object code.

The following lines of high level language code are compiled.

```
X = X + Y
Z = Z + X
```

The compilation produces the assembly language code as follows:

```
LDD 236
ADD 237
STO 236
LDD 238
ADD 236
STO 238
```

(i) The final stage in the compilation process that follows this code generation stage is code optimisation.

Rewrite the equivalent code after optimisation.

....

....

....

....

....

.... [3]

(ii) Explain why code optimisation is necessary.

....

....

....

.... [2]