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**	
LDM	#n	Immediate addressing. Load the number n to ACC.
LDD	<address>	Direct addressing. Load the contents of the location at the given address to ACC.
LDI	<address>	Indirect addressing. The address to be used is at the given address. Load the contents of this second address to ACC.
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).
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.
JPN	<address>	Following a compare instruction, jump to <address> if the compare was False.
AND	<address>	Bitwise AND operation of the contents of ACC with the contents of <address>.
XOR	<address>	Bitwise XOR operation of the contents of ACC with the contents of <address>.
OR	<address>	Bitwise OR operation of the contents of ACC with the contents of <address>.
IN		Key in a character and store its ASCII value in ACC.
OUT		Output to the screen the character whose ASCII value is stored in ACC.
END		Return control to the operating system.

(a) A programmer writes a program that:

- reads two characters input from the keyboard (you may assume they will be capital letters in ascending alphabetical sequence)
- outputs the alphabetical sequence of characters from the first to the second character. For example, if the characters ‘B’ and ‘F’ are input, the output is:

  BCDEF

The programmer has started to write the program in the following table. The Comment column contains descriptions for the missing program instructions, labels and data.

Complete the following program. Use op codes from the given instruction set.

Label	Op code	Operand	Comment
START:			// INPUT character
			// store in CHAR1
			// INPUT character
			// store in CHAR2
			// initialise ACC to ASCII value of CHAR1
			// output contents of ACC
			// compare ACC with CHAR2
			// if equal jump to end of FOR loop
			// increment ACC
			// jump to LOOP
ENDFOR:	END		
CHAR1:			
CHAR2:			

[9]

(b) The programmer now starts to write a program that:

- converts a positive integer, stored at address NUMBER1, into its negative equivalent in two's complement form
- stores the result at address NUMBER2

Complete the following program. Use op codes from the given instruction set.
Show the value stored in NUMBER2.

Label	Op code	Operand	Comment
START:			
		MASK	// convert to one's complement
			// convert to two's complement
	END		
MASK:			// show value of mask in binary here
NUMBER1:	B00000101		// positive integer
NUMBER2:			// negative equivalent

[6]

8 The table shows assembly language instructions for a processor that has one general purpose register, the Accumulator (ACC), and an Index Register (IX).

	Instruction		Explanation
Label	**Op code**	**Operand**	
	LDM	#n	Immediate addressing. Load the number n to ACC
	LDD	<address>	Direct addressing. Load the contents of the location at the given address to ACC
	LDX	<address>	Indexed addressing. Form the address from <address> + the contents of the Index Register. Copy the contents of this calculated address to ACC
	LDR	#n	Immediate addressing. Load the number n to IX
	STO	<address>	Store the contents of ACC at the given address
	ADD	<address>	Add the contents of the given address to ACC
	INC	<register>	Add 1 to the contents of the register (ACC or IX)
	CMP	#n	Compare the contents of ACC with number n
	JPN	<address>	Following a compare instruction, jump to <address> if the compare was False
	LSL	#n	Bits in ACC are shifted n places to the left. Zeroes are introduced on the right hand end
	LSR	#n	Bits in ACC are shifted n places to the right. Zeroes are introduced on the left hand end
	OUT		Output to the screen the character whose ASCII value is stored in ACC
	END		Return control to the operating system
<label>:	<op code>	<operand>	Labels an instruction
<label>:	<data>		Gives a symbolic address <label> to the memory location with contents <data>

An algorithm stores a 3-character word. It takes each character in turn, multiplies its value by 2 and outputs the new character.

Complete the following assembly language program for the algorithm using the instruction set provided on the previous page.

Instruction			Comment
Label	**Op code**	**Operand**	
			// initialise Index Register to 0
	LDX	character	// load character and multiply by 2
	OUT		// output the new character
	INC	IX	// increment the Index Register
	LDD	count	// loop 3 times
	STO	count	
	CMP	#3	
		LOOP	
	END		// end program
count:	0		
character:	B01000001		// the 3-character stored word
	B10001110		
	B01000100		

[5]

5 The table shows assembly language instructions for a processor which has one general purpose register – the Accumulator (ACC).

Instruction		Explanation
Op Code	**Operand**	
LDM	#n	Immediate addressing. Load the number n to ACC
LDD	<address>	Direct addressing. Load the contents of the given address to ACC
STO	<address>	Store the contents of ACC at the given address
ADD	<address>	Add the contents of the given address to the ACC
INC	<register>	Add 1 to the contents of the register
CMP	<address>	Compare the contents of ACC with the contents of <address>
JPN	<address>	Following a compare instruction, jump to <address> if the compare was False
END		Return control to the operating system

(a) (i) Dry-run this assembly language program using the trace table.

500	LDD 512
501	ADD 509
502	STO 512
503	LDD 511
504	INC ACC
505	STO 511
506	CMP 510
507	JPN 500
508	END
509	7
510	3
511	0
512	0

Trace table

	Memory address			
Accumulator	**509**	**510**	**511**	**512**
	7	3	0	0

[5]

(ii) Explain the role address 511 has in this assembly language program.

.....

.....

.....

..... [2]

(b) Using opcodes from the given table, write instructions to set the value at address 509 to 12.

.....

.....

.....

..... [2]

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**	
LDM	#n	Immediate addressing. Load the number n to ACC.
LDD	<address>	Direct addressing. Load the contents of the given address to ACC.
LDX	<address>	Indexed addressing. Form the address from <address> + the contents of the index register. Copy the contents of this calculated address to ACC.
LDR	#n	Immediate addressing. Load the number n into IX.
STO	<address>	Store the contents of ACC at the given address.
INC	<register>	Add 1 to the contents of the register (ACC or IX).
CMP	<address>	Compare the contents of ACC with the contents of <address>.
CMP	#n	Compare the contents of ACC with number n.
JPE	<address>	Following a compare instruction, jump to <address> if the compare was True.
JPN	<address>	Following a compare instruction, jump to <address> if the compare was False.
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>.
OUT		Output to the screen the character whose ASCII value is stored in ACC.
END		Return control to the operating system.

A programmer is writing a program that will output the first character of a string in upper case and the remaining characters of the string in lower case.

The program will use locations from address `WORD` onwards to store the characters in the string. The location with address `LENGTH` stores the number of characters that make up the string.

The programmer has started to write the program in the following table. The comment column contains descriptions for the missing program instructions.

(b) Complete the program using op codes from the given instruction set.

Label	Op code	Operand	Comment
START:			// initialise index register to zero
			// get first character of WORD
			// ensure it is in upper case using MASK1
			// output character to screen
			// increment index register
			// load 1 into ACC
			// store in COUNT
LOOP:			// load next character from indexed address WORD
			// make lower case using MASK2
			// output character to screen
			// increment COUNT starts here
			// is COUNT = LENGTH ?
			// if FALSE, jump to LOOP
			// end of program
COUNT:			
MASK1:			// bit pattern for upper case
MASK2:			// bit pattern for lower case
LENGTH:		4	
WORD:		B01100110	// ASCII code in binary for 'f'
		B01110010	// ASCII code in binary for 'r'
		B01000101	// ASCII code in binary for 'E'
		B01000100	// ASCII code in binary for 'D'

[12]

[Turn over

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

Instruction		Explanation
Op code	**Operand**	
LDM	#n	Immediate addressing. Load the number n to ACC.
LDD	<address>	Direct addressing. Load the contents of the location at the given address to ACC.
LDI	<address>	Indirect addressing. The address to be used is at the given address. Load the contents of this second address to ACC.
LDX	<address>	Indexed addressing. Form the address from <address> + the contents of the index register. Copy the contents of this calculated address to ACC.
LDR	#n	Immediate addressing. Load the number n to IX.
STO	<address>	Store the contents of ACC at the given address.
STX	<address>	Indexed addressing. Form the address from <address> + the contents of the index register. Copy the contents from ACC to this calculated address.
ADD	<address>	Add the contents of the given address to the ACC.
INC	<register>	Add 1 to the contents of the register (ACC or IX).
DEC	<register>	Subtract 1 from the contents of the register (ACC or IX).
JMP	<address>	Jump to the given address.
CMP	<address>	Compare the contents of ACC with the contents of <address>.
CMP	#n	Compare the contents of ACC with number n.
JPE	<address>	Following a compare instruction, jump to <address> if the compare was True.
JPN	<address>	Following a compare instruction, jump to <address> if the compare was False.
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.
IN		Key in a character and store its ASCII value in ACC.
OUT		Output to the screen the character whose ASCII value is stored in ACC.
END		Return control to the operating system.

(a) Six letters are stored, starting at the location labelled LETTERS. A program is needed to perform a linear search on LETTERS to find the letter 'x'. The program counts the number of times 'x' appears in LETTERS.

The following is the pseudocode for the program.

```
FOR COUNT ← 0 TO 5
   IF LETTERS[COUNT] = LETTERTOFIND
      THEN
         FOUND ← FOUND + 1
   ENDIF
ENDFOR
```

Write this program. Use the op codes from the given instruction set.

Label	Op code	Operand	Comment
START:	LDR	#0	// initialise Index Register
LOOP:			// load LETTERS
			// is LETTERS = LETTERTOFIND ?
			// if not, go to NOTFOUND
			// increment FOUND
NOTFOUND:			// increment COUNT
			// is COUNT = 6 ?
			// if yes, end
			// increment Index Register
			// go back to beginning of loop
ENDP:	END		// end program
LETTERTOFIND:	'x'		
LETTERS:	'd'		
	'u'		
	'p'		
	'l'		
	'e'		
	'x'		
COUNT:	0		
FOUND:	0		

[10]

(b) Six values are stored, starting at the location `VALUES`. A program is needed to divide each of the values by 8 and store them back in their original location.

Write this program. Use the op codes from the instruction set on the next page.

Label	Op code	Operand	Comment
START:			// initialise the Index Register
			// load the value from VALUES
			// divide by 8
			// store the new value in VALUES
			// increment the Index Register
			// increment REPS
			// is REPS = 6 ?
			// repeat for next value
	END		
REPS:	0		
VALUES:	22		
	13		
	5		
	46		
	12		
	33		

[10]

Instruction		Explanation
Op code	**Operand**	
LDM	#n	Immediate addressing. Load the number n to ACC.
LDD	<address>	Direct addressing. Load the contents of the location at the given address to ACC.
LDI	<address>	Indirect addressing. The address to be used is at the given address. Load the contents of this second address to ACC.
LDX	<address>	Indexed addressing. Form the address from <address> + the contents of the index register. Copy the contents of this calculated address to ACC.
LDR	#n	Immediate addressing. Load the number n to IX.
STO	<address>	Store the contents of ACC at the given address.
STX	<address>	Indexed addressing. Form the address from <address> + the contents of the index register. Copy the contents from ACC to this calculated address.
ADD	<address>	Add the contents of the given address to the ACC.
INC	<register>	Add 1 to the contents of the register (ACC or IX).
DEC	<register>	Subtract 1 from the contents of the register (ACC or IX).
JMP	<address>	Jump to the given address.
CMP	<address>	Compare the contents of ACC with the contents of <address>.
CMP	#n	Compare the contents of ACC with number n.
JPE	<address>	Following a compare instruction, jump to <address> if the compare was True.
JPN	<address>	Following a compare instruction, jump to <address> if the compare was False.
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.
IN		Key in a character and store its ASCII value in ACC.
OUT		Output to the screen the character whose ASCII value is stored in ACC.
END		Return control to the operating system.

[Turn over

5 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**	
LDM	#n	Immediate addressing. Load the number n to ACC.
LDD	<address>	Direct addressing. Load the contents of the location at the given address to ACC.
LDI	<address>	Indirect addressing. The address to be used is at the given address. Load the contents of this second address to ACC.
LDX	<address>	Indexed addressing. Form the address from <address> + the contents of the Index Register. Copy the contents of this calculated address to ACC.
LDR	#n	Immediate addressing. Load the number n to IX.
STO	<address>	Store the contents of ACC at the given address.
STX	<address>	Indexed addressing. Form the address from <address> + the contents of the Index Register. Copy the contents from ACC to this calculated address.
ADD	<address>	Add the contents of the given address to the ACC.
INC	<register>	Add 1 to the contents of the register (ACC or IX).
DEC	<register>	Subtract 1 from the contents of the register (ACC or IX).
JMP	<address>	Jump to the given address.
CMP	<address>	Compare the contents of ACC with the contents of <address>.
CMP	#n	Compare the contents of ACC with number n.
JPE	<address>	Following a compare instruction, jump to <address> if the compare was True.
JPN	<address>	Following a compare instruction, jump to <address> if the compare was False.
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.
IN		Key in a character and store its ASCII value in ACC.
OUT		Output to the screen the character whose ASCII value is stored in ACC.
END		Return control to the operating system.

(a) A programmer needs a program that multiplies a binary number by 4.

The programmer has started to write the program in the following table. The comment column contains explanations for the missing program instructions.

Write the program using the given instruction set.

Label	Instruction		Comment
	Op code	Operand	
			`// load contents of NUMBER`
			`// perform shift to multiply by 4`
			`// store contents of ACC in NUMBER`
			`// end program`
`NUMBER:`	`B00110110`		

[5]

Note:

`#` denotes immediate addressing
`B` denotes a binary number, e.g. `B01001010`
`&` denotes a hexadecimal number, e.g. `&4A`

[Turn over

(b) A programmer needs a program that counts the number of lower case letters in a string.

The programmer has started to write the program in the following table. The comment column contains explanations for the missing program instructions.

Complete the program using the given instruction set. A copy of the instruction set is provided on the opposite page.

Label	Instruction		Comment
	Op code	Operand	
	LDR	#0	// initialise Index Register to 0
START:			// load the next value from the STRING
			// perform bitwise AND operation with MASK
			// check if result is equal to MASK
			// if FALSE, jump to UPPER
			// increment COUNT
UPPER:	INC	IX	// increment the Index Register
			// decrement LENGTH
			// is LENGTH = 0 ?
			// if FALSE, jump to START
	END		// end program
MASK:	B00100000		// if bit 5 is 1, letter is lower case
COUNT:	0		
LENGTH:	5		
STRING:	B01001000		// ASCII code for 'H'
	B01100001		// ASCII code for 'a'
	B01110000		// ASCII code for 'p'
	B01110000		// ASCII code for 'p'
	B01011001		// ASCII code for 'Y'

[8]

Instruction		Explanation
Op code	**Operand**	
LDM	#n	Immediate addressing. Load the number n to ACC.
LDD	<address>	Direct addressing. Load the contents of the location at the given address to ACC.
LDI	<address>	Indirect addressing. The address to be used is at the given address. Load the contents of this second address to ACC.
LDX	<address>	Indexed addressing. Form the address from <address> + the contents of the Index Register. Copy the contents of this calculated address to ACC.
LDR	#n	Immediate addressing. Load the number n to IX.
STO	<address>	Store the contents of ACC at the given address.
STX	<address>	Indexed addressing. Form the address from <address> + the contents of the Index Register. Copy the contents from ACC to this calculated address.
ADD	<address>	Add the contents of the given address to the ACC.
INC	<register>	Add 1 to the contents of the register (ACC or IX).
DEC	<register>	Subtract 1 from the contents of the register (ACC or IX).
JMP	<address>	Jump to the given address.
CMP	<address>	Compare the contents of ACC with the contents of <address>.
CMP	#n	Compare the contents of ACC with number n.
JPE	<address>	Following a compare instruction, jump to <address> if the compare was True.
JPN	<address>	Following a compare instruction, jump to <address> if the compare was False.
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.
IN		Key in a character and store its ASCII value in ACC.
OUT		Output to the screen the character whose ASCII value is stored in ACC.
END		Return control to the operating system.

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

Instruction		Explanation
Op code	**Operand**	
LDM	#n	Immediate addressing. Load the number n to ACC.
LDD	<address>	Direct addressing. Load the contents of the location at the given address to ACC.
LDI	<address>	Indirect addressing. The address to be used is at the given address. Load the contents of this second address to ACC.
LDX	<address>	Indexed addressing. Form the address from <address> + the contents of the Index Register. Copy the contents of this calculated address to ACC.
LDR	#n	Immediate addressing. Load the number n to IX.
STO	<address>	Store the contents of ACC at the given address.
STX	<address>	Indexed addressing. Form the address from <address> + the contents of the Index Register. Copy the contents of ACC to this calculated address.
ADD	<address>	Add the contents of the given address to ACC.
INC	<register>	Add 1 to the contents of the register (ACC or IX).
DEC	<register>	Subtract 1 from the contents of the register (ACC or IX).
JMP	<address>	Jump to the given address.
CMP	<address>	Compare the contents of ACC with the contents of <address>.
CMP	#n	Compare the contents of ACC with number n.
JPE	<address>	Following a compare instruction, jump to <address> if the compare was True.
JPN	<address>	Following a compare instruction, jump to <address> if the compare was False.
IN		Key in a character and store its ASCII value in ACC.
OUT		Output to the screen the character whose ASCII value is stored in ACC.
END		Return control to the operating system.

Consider the following pseudocode algorithm:

```
Length ← 0
INPUT Character
WHILE Character <> "."
    Message ← Message & Character
    Length ← Length + 1
    INPUT Character
ENDWHILE
```

Complete the table by writing assembly language code for the algorithm, using the given instruction set.

Label	Instruction		Comment
	Op code	Operand	
			// initialise IX to zero
			// initialise LENGTH
LOOP:			// input character
			// is character a FULLSTOP (.) ?
			// jump to ENDP if TRUE
			// store character in MESSAGE + contents of IX
			// increment IX
			// increment LENGTH
			// jump to LOOP
ENDP:	END		// end program
LENGTH:			
FULLSTOP:	B01100000		// ASCII code for a full stop (.)
MESSAGE:			

[8]