QUESTION 8.

7 The table shows assembly language instructions for a processor which has one
register, the Accumulator (ACC).

Instruction		Explanation
Op code	**Operand**	
LDD	<address>	Direct addressing. Load contents of given address to ACC
STO	<address>	Store the contents of ACC at the given address
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
INC	<register>	Add 1 to contents of the register (ACC)
JMP	<address>	Jump to the given address
END		Return control to operating system

The diagram shows the contents of the memory.

Main memory

120	0000 1001
121	0111 0101
122	1011 0110
123	1110 0100
124	0111 1111
125	0000 0001
126	0100 0001
127	0110 1001
200	1000 1000

(a) **(i)** Show the contents of the Accumulator after execution of the instruction:

LDD 121

Accumulator:								

[1]

(ii) Show the contents of the Accumulator after execution of the instruction:

LDI 124

Accumulator:								

Explain how you arrived at your answer.

....

....

....

.... [3]

(iii) Show the contents of the Accumulator after execution of the instruction:

LDX 120

Index Register:	0	0	0	0	0	1	1	0

Accumulator:								

Explain how you arrived at your answer.

....

....

....

.... [3]

QUESTION 9.

9 The table shows assembly language instructions for a processor which has one
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.
`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 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).
`DEC`	`<register>`	Subtract 1 from the contents of the register (ACC or IX).
`CMP`	`<address>`	Compare contents of ACC with contents of <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.
`JMP`	`<address>`	Jump to the given address.
`OUT`		Output to screen the character whose ASCII value is stored in ACC.
`END`		Return control to the operating system.

(a) The diagram shows the current contents of a section of main memory and the index register:

60	0011 0010
61	0101 1101
62	0000 0100
63	1111 1001
64	0101 0101
65	1101 1111
66	0000 1101
67	0100 1101
68	0100 0101
69	0100 0011
...	
1000	0110 1001

Index register:

0	0	0	0	1	0	0	0

(i) Show the contents of the Accumulator after the execution of the instructi

```
LDX 60
```

Accumulator:								

Show how you obtained your answer.

....

....

....

....[2]

(ii) Show the contents of the index register after the execution of the instruction:

```
DEC IX
```

Index register:								

[1]

(b) Complete the trace table on the opposite page for the following assembly lan

50	LDD 100
51	ADD 102
52	STO 103
53	LDX 100
54	ADD 100
55	CMP 101
56	JPE 58
57	JPN 59
58	OUT
59	INC IX
60	LDX 98
61	ADD 101
62	OUT
63	END
...	
100	20
101	100
102	1
103	0

IX (Index Register)	1

Selected values from the ASCII character set:

ASCII Code	118	119	120	121	122	123	124	125
Character	v	w	x	y	z	{	|	}

Trace table:

Instruction address	Working space	ACC	Memory address				IX	OUTPUT
			100	101	102	103		
			20	100	1	0	1	
50								
51								
52								
53								
54								
55								

[7]

QUESTION 10.

4 The table shows assembly language instructions for a processor which has one
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.
`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 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).
`DEC`	`<register>`	Subtract 1 from the contents of the register (ACC or IX).
`CMP`	`<address>`	Compare contents of ACC with contents of <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.
`JMP`	`<address>`	Jump to the given address.
`OUT`		Output to screen the character whose ASCII value is stored in ACC.
`END`		Return control to the operating system.

The diagram shows the contents of the index register:

Index register:								
	1	1	0	0	1	1	0	1

(a) Show the contents of the index register after the execution of the instruction:

```
INC IX
```

Index register:								

[1]

(b) Complete the trace table on the opposite page for the following assembly lan

20	`LDX 90`
21	`DEC ACC`
22	`STO 90`
23	`INC IX`
24	`LDX 90`
25	`DEC ACC`
26	`CMP 90`
27	`JPE 29`
28	`JPN 31`
29	`ADD 90`
30	`OUT`
31	`ADD 93`
32	`STO 93`
33	`OUT`
34	`END`
:	
90	`2`
91	`90`
92	`55`
93	`34`

IX	`2`

Selected values from the ASCII character set:

ASCII Code	65	66	67	68	69	70	71	72
Character	A	B	C	D	E	F	G	H

Trace table:

Instruction	Working space	ACC	Memory address				IX	OUTPUT
			90	91	92	93		
			2	90	55	34	2	
20								
21								
22								
23								
24								
25								
26								

[7]

QUESTION 11.

8 The table shows assembly language instructions for a processor which has one g
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.
`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 contents of ACC at the given address.
`ADD`	`<address>`	Add the contents of the given address to ACC.
`CMP`	`<address>`	Compare contents of ACC with contents of `<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.
`JMP`	`<address>`	Jump to the given address.
`OUT`		Output to the screen the character whose ASCII value is stored in ACC.
`END`		Return control to the operating system.

The diagram shows the contents of the main memory:

Main memory

800	0110 0100
801	0111 1100
802	1001 0111
803	0111 0011
804	1001 0000
805	0011 1111
806	0000 1110
807	1110 1000
808	1000 1110
809	1100 0010
:	
:	
2000	1011 0101

(a) **(i)** Show the contents of the Accumulator after execution of the instruction:

`LDD 802`

Accumulator:								

[1]

(ii) Show the contents of the Accumulator after execution of the instruction:

```
LDX 800
```

Index Register:	0	0	0	0	1	0	0	1
Accumulator:								

Explain how you arrived at your answer.

....

....

....

....[3]

(b) **(i)** Complete the trace table below for the following assembly languag
program contains denary values.

100	`LDD 800`
101	`ADD 801`
102	`STO 802`
103	`LDD 803`
104	`CMP 802`
105	`JPE 107`
106	`JPN 110`
107	`STO 802`
108	`OUT`
109	`JMP 112`
110	`LDD 801`
111	`OUT`
112	`END`
⋮	
800	`40`
801	`50`
802	`0`
803	`90`

Selected values from the ASCII character set:

ASCII code	40	50	80	90	100
Character	(	2	P	Z	d

Trace table:

ACC	**Memory address**				**OUTPUT**
	800	**801**	**802**	**803**	
	40	50	0	90	

[4]

(ii) There is a redundant instruction in the code in **part (b)(i)**.

State the address of this instruction.

....

(c) The program used the ASCII coding system for character codes. An alternative coding system is Unicode.

(i) Give **two** disadvantages of using ASCII code.

1 ....

....

2 ....

....[2]

(ii) Describe how Unicode is designed to overcome the disadvantages of ASCII.

....

....

....

....[2]

BLANK PAGE

BLANK PAGE

QUESTION 12.

5 The table shows assembly language instructions for a processor that has one 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.
`LDX`	`<address>`	Index addressing. Form the address from `<address>` + the contents of the index register. Copy the contents of this calculated 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.
`STO`	`<address>`	Store the contents of ACC at the given address.
`INC`	`<register>`	Add 1 to contents of the register (ACC or IX).
`ADD`	`<address>`	Add the contents of the given address to the ACC.
`END`		Return control to the operating system.

The diagram shows the contents of a section of main memory:

Main memory

100	`0000 0010`
101	`1001 0011`
102	`0111 0011`
103	`0110 1011`
104	`0111 1110`
105	`1011 0001`
106	`0110 1000`
107	`0100 1011`
…	
200	`1001 1110`

(a) **(i)** Show the contents of the Accumulator after the execution of the instructi

`LDD 102`

ACC:								

[1]

(ii) Show the contents of the Accumulator after the execution of the instruction:

`LDX 101`

IX:	0	0	0	0	0	1	0	0

ACC:								

Explain how you arrived at your answer.

....

....

....

....[2]

(iii) Show the contents of the Accumulator after the execution of the instruction:

`LDI 103`

ACC:								

Explain how you arrived at your answer.

....

....

....

....

....

....[3]

(b) Complete the trace table below for the following assembly language program

800	`LDD 810`
801	`INC ACC`
802	`STO 812`
803	`LDD 811`
804	`ADD 812`
805	`STO 813`
806	`END`
…	
810	`28`
811	`41`
812	`0`
813	`0`

Trace table:

ACC	**Memory address**			
	810	811	812	813
	28	41	0	0

[6]

QUESTION 13.

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

Instruction			
Op code (mnemonic)	**Operand**	**Op code (binary)**	**Explanation**
`LDM`	`#n`	0000 0001	Immediate addressing. Load the denary number `n` to ACC.
`LDD`	`<address>`	0000 0010	Direct addressing. Load the contents of the location at the given address to ACC.
`LDI`	`<address>`	0000 0101	Indirect addressing. At the given address is the address to be used. Load the contents of this second address to ACC.
`LDX`	`<address>`	0000 0110	Indexed addressing. Form the address from `<address>` + the contents of the Index Register (IX). Copy the contents of this calculated address to ACC.
`LDR`	`#n`	0000 0111	Immediate addressing. Load number `n` to IX.
`STO`	`<address>`	0000 1111	Store the contents of ACC at the given address.

The following diagram shows the contents of a section of main memory and the Index Register (IX).

(a) Show the contents of the Accumulator (ACC) after each instruction is executed.

IX	0	0	0	0	0	0	1	1

(i) `LDM #500`

ACC ....[1]

(ii) `LDD 500`

ACC ....[1]

(iii) `LDX 500`

ACC ....[1]

(iv) `LDI 500`

ACC ....[1]

Address	Main Memory contents
495	13
496	86
497	92
498	486
499	489
500	496
501	497
502	499
503	502

(b) Each machine code instruction is encoded as 16-bits (8-bit op code follo operand).

Write the machine code for the following instructions:

`LDM #17`

`LDX #97`

[3]

(c) Using an 8-bit operand, state the maximum number of memory locations, in denary, that can be directly addressed.

....[1]

(d) Computer scientists often write binary representations in hexadecimal.

(i) Write the hexadecimal representation for this instruction:

0	0	0	0	0	1	1	1		1	1	0	0	0	0	1	0

....[2]

(ii) A second instruction has been written in hexadecimal as:

`05 3F`

Write the equivalent assembly language instruction, with the operand in denary.

....[2]

QUESTION 14.

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

Instruction			Explanation
Op code (mnemonic)	**Operand**	**Op code (binary)**	
`LDD`	`<address>`	`0001 0011`	Direct addressing. Load the contents of the location at the given address to the Accumulator (ACC).
`LDI`	`<address>`	`0001 0100`	Indirect addressing. The address to be used is at the given address. Load the contents of this second address to ACC.
`LDX`	`<address>`	`0001 0101`	Indexed addressing. Form the address from `<address>` + the contents of the Index Register. Copy the contents of this calculated address to ACC.
`LDM`	`#n`	`0001 0010`	Immediate addressing. Load the denary number `n` to ACC.
`LDR`	`#n`	`0001 0110`	Immediate addressing. Load denary number `n` to the Index Register (IX).
`STO`	`<address>`	`0000 0111`	Store the contents of ACC at the given address.

The following diagram shows the contents of a section of main memory and the Index Register (IX).

(a) Show the contents of the Accumulator (ACC) after each instruction is executed.

IX	0	0	0	0	0	1	1	0

(i) `LDD 355`

ACC .... [1]

(ii) `LDM #355`

ACC .... [1]

(iii) `LDX 351`

ACC .... [1]

(iv) `LDI 355`

ACC .... [1]

Address	Main memory contents
350	
351	86
352	
353	
354	
355	351
356	
357	22
358	

(b) Each machine code instruction is encoded as 16 bits (8-bit op code follo operand).

Write the machine code for these instructions:

`LDM #67`

`LDX #7`

[3]

(c) Computer scientists often write binary representations in hexadecimal.

(i) Write the hexadecimal representation for the following instruction.

0	0	0	1	0	1	0	0	0	1	0	1	1	1	1	0

....[2]

(ii) A second instruction has been written in hexadecimal as:

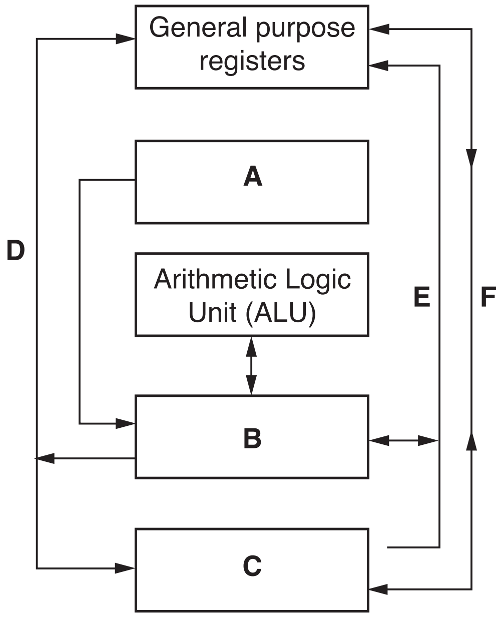
`16 4D`

Write the assembly language for this instruction with the operand in denary.

....[2]

QUESTION 15.

4 **(a)** The diagram shows the components and buses found inside a typical Pe (PC).



Some components and buses only have labels **A** to **F** to identify them.

For each label, choose the appropriate title from the following list. The title for label **D** is already given.

- Control bus
- System clock
- Data bus
- Control unit
- Main memory
- Secondary storage

A ....

B ....

C ....

D Address bus

E ....

F ....

[5]

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

Instruction			Explanation
Op code (mnemonic)	**Operand**	**Op code (binary)**	
`LDM`	`#n`	`1100 0001`	Immediate addressing. Load number `n` to ACC.
`LDD`	`<address>`	`1100 0010`	Direct addressing. Load the contents of the given address to ACC.
`LDV`	`#n`	`1100 0011`	Relative addressing. Move to the address `n` locations from the address of the current instruction. Load the contents of this address to ACC.
`STO`	`<address>`	`1100 0100`	Store the contents of ACC at the given address.
`DEC`		`1100 0101`	Decrement the contents of ACC.
`OUTCH`		`1100 0111`	Output the character corresponding to the ASCII character code in ACC.
`JNE`	`<address>`	`1110 0110`	Following a compare instruction, jump to `<address>` if the compare was False.
`JMP`	`<address>`	`1110 0011`	(Unconditionally) jump to the given address.
`CMP`	`#n`	`1110 0100`	Compare the contents of ACC with number `n`.

Complete the trace table for the following assembly language program.

Label	Instruction
StartProg:	LDV #CountDown
	CMP Num1
	JNE CarryOn
	JMP Finish
CarryOn:	OUTCH
	LDD CountDown
	DEC
	STO CountDown
	JMP StartProg
Finish:	LDM #88
	OUTCH
	END
CountDown:	15
	32
	51
	67
Num1:	32

ASCII code table (selected codes only)				
<Space>	3	B	C	X
32	51	66	67	88

Trace table:

ACC	CountDown	OUTPUT
	15	
67		C
15		

[5]

(c) The program given in **part (b)** is to be translated using a two-pass assemble

The program has been copied here for you. The program now starts with a dire tells the assembler to load the first instruction of the program to address 100.

Label	
	`ORG    #0100`
`StartProg:`	`LDV    #CountDown`
	`CMP    Num1`
	`JNE    CarryOn`
	`JMP    Finish`
`CarryOn:`	`OUTCH`
	`LDD    CountDown`
	`DEC`
	`STO    CountDown`
	`JMP    StartProg`
`Finish:`	`LDM    #88`
	`OUTCH`
	`END`
`CountDown:`	`15`
	`32`
	`51`
	`67`
`Num1:`	`32`

On the first pass of the two-pass process, the assembler adds entries to a sy

The following symbol table shows the first eleven entries, part way through the firs

The circular labels show the order in which the assembler made the entries to the sy table.

Symbol table

Symbolic address		Absolute address			
`StartProg`	1	100	2		
`CountDown`	3	UNKNOWN	4		
`Num1`	5	UNKNOWN	6		
`CarryOn`	7	~~UNKNOWN~~	8	104	11
`Finish`	9	UNKNOWN	10		

Explain how the assembler made these entries to the symbol table.

....

....

....

....

....

....[3]

(d) The assembler software must then complete the second pass building up the executable file.

(i) Name the second table needed when the assembler software carries out the second pass.

....[1]

The following shows two of the program instructions in machine code.

Instruction	Machine code	
	Binary	**Hexadecimal**
`OUTCH`	`1100 0111`	`C7`
`JNE CarryOn`	**A**	**B**

Each of the numbers **A** and **B** represents the complete instruction in two bytes, one byte for the op code and one byte for the operand.

(ii) Use the following instruction set to write the numbers for **A** and **B**.

A (binary) ....

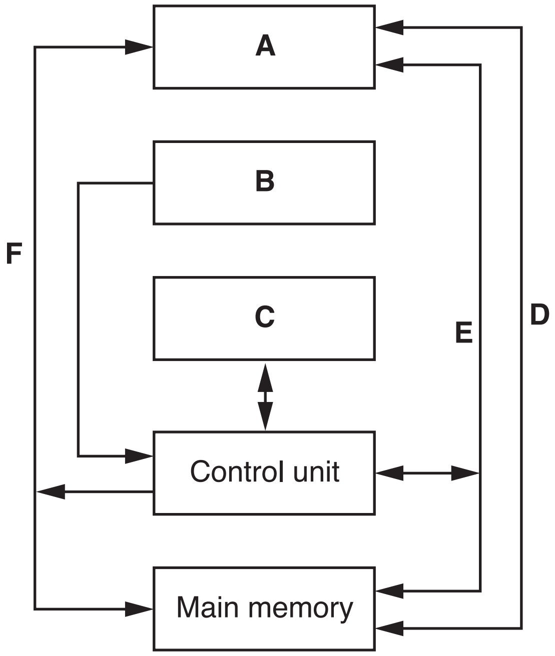
B (hexadecimal) ....

[3]

Instruction			Explanation
Op code (mnemonic)	**Operand**	**Op code (binary)**	
`LDM`	`#n`	`1100 0001`	Immediate addressing. Load number `n` to ACC.
`LDD`	`<address>`	`1100 0010`	Direct addressing. Load the contents of the given address to ACC.
`LDV`	`#n`	`1100 0011`	Relative addressing. Move to the address `n` locations from the address of the current instruction. Load the contents of this address to ACC.
`STO`	`<address>`	`1100 0100`	Store the contents of ACC at the given address.
`DEC`		`1100 0101`	Decrement the contents of ACC.
`OUTCH`		`1100 0111`	Output the character corresponding to the ASCII character code in ACC.
`JNE`	`<address>`	`1110 0110`	Following a compare instruction, jump to `<address>` if the compare was False.
`JMP`	`<address>`	`1110 0011`	(Unconditionally) jump to the given address.
`CMP`	`#n`	`1110 0100`	Compare the contents of ACC with number `n`.

QUESTION 16.

4 The following diagram shows the components and buses found inside a typical pe
(PC).



(a) Some components and buses only have labels **A** to **F** to identify them.

For each label, choose the appropriate title from the following list. The title for label **D** is already given.

- Control bus
- Address bus
- Arithmetic Logic Unit (ALU)
- General purpose registers
- Secondary storage
- System clock

A ....

B ....

C ....

D Data bus

E ....

F ....
[5]

(b) Clock speed is a factor that affects the performance of a PC. Explain this statement.

....

....

....[2]

(c) An assembly language program can contain both **macros** and **directives**.

(i) Explain what is meant by these terms.

Macro ....

....

....

Directive ....

....

....
[3]

(ii) Give an example of the use of a directive.

....

....[1]

(d) 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 (mnemonic)	**Operand**	
`LDD`	`<address>`	Direct addressing. Load the contents of the given address to ACC.
`LDV`	`#n`	Relative addressing. Move to the address `n` locations from the address of the current instruction. Load the contents of this address to ACC.
`STO`	`<address>`	Store the contents of ACC at the given address.
`INC`		Increment the contents of ACC.
`OUTCH`		Output the character corresponding to the ASCII character code in ACC.
`JPE`	`<address>`	Following a compare instruction, jump to `<address>` if the compare was True.
`JMP`	`<address>`	Jump to the given address.
`CMP`	`#n`	Compare the contents of ACC with number `n`.

Complete the trace table for the following assembly language program.

Label	Instruction
`StartProg:`	`LDV    #Offset`
	`CMP    Value`
	`JPE    EndProg`
	`OUTCH`
	`LDD    Offset`
	`INC`
	`STO    Offset`
	`JMP    StartProg`
`EndProg:`	`END`
`Offset:`	`10`
	`50`
	`65`
	`89`
	`32`
`Value:`	`32`

ASCII code table (selected codes only)				
<Space>	2	A	B	Y
32	50	65	66	89

Trace table:

ACC	`Offset`	OUTPUT
	`10`	
`50`		`2`
`10`		

[5]

(e) The program given in **part (d)** is to be translated using a two-pass assembl has been copied here for you.

Label	**Instruction**
`StartProg:`	`LDV    #Offset`
	`CMP    Value`
	`JPE    EndProg`
	`OUTCH`
	`LDD    Offset`
	`INC`
	`STO    Offset`
	`JMP    StartProg`
`EndProg:`	`END`
`Offset:`	`10`
	`50`
	`65`
	`89`
	`32`
`Value:`	`32`

On the first pass, the assembly process adds entries to a symbol table.

The following symbol table shows the first five entries, part way through the first pass.

The circular labels show the order in which the assembler made the entries to the symbol table.

Complete the symbol table. Use circular labels to show the order in which the assembler makes the entries.

Symbol table

Symbolic address	**Relative address**
`StartProg` ①	0 ②
`Offset` ③	UNKNOWN ④
`Value` ⑤	

[6]

QUESTION 17.

4 The table shows assembly language instructions for a processor which has one register, the Accumulator (ACC) and an index register (IX).

Instruction		Explanation
Op code	Operand	
`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 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).
`DEC`	`<register>`	Subtract 1 from the contents of the register (ACC or IX).
`CMP`	`<address>`	Compare contents of ACC with contents of `<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.
`JMP`	`<address>`	Jump to the given address.
`OUT`		Output to the screen the character whose ASCII value is stored in ACC.
`END`		Return control to the operating system.

(a) (i) State what is meant by **direct addressing** and **indirect addressing**.

Direct addressing ....

....

Indirect addressing ....

.... [2]

(ii) Explain how the instruction `ADD 20` can be interpreted as either direct or indirect addressing.

Direct addressing ....

....

Indirect addressing ....

.... [2]

(b) The assembly language instructions in the following table use either symbo absolute addressing.

Tick (✓) **one** box in each row to indicate whether the instruction uses symbolic or addressing.

Instruction	Symbolic	Absolute
`ADD 90`		
`CMP found`		
`STO 20`		

[2]

(c) The current contents of a general purpose register (X) are:

X	1	0	1	1	1	0	1	0

(i) The contents of X represent an unsigned binary integer.

Convert the value in X into denary.

....[1]

(ii) The contents of X represent an unsigned binary integer.

Convert the value in X into hexadecimal.

....[1]

(iii) The contents of X represent a two's complement binary integer.

Convert the value in X into denary.

....[1]

(d) The current contents of the main memory, Index Register (IX) and selected ASCII character set are provided with a copy of the instruction set.

Address	Instruction
70	`LDX 200`
71	`OUT`
72	`STO 203`
73	`LDD 204`
74	`INC ACC`
75	`STO 204`
76	`INC IX`
77	`LDX 200`
78	`CMP 203`
79	`JPN 81`
80	`OUT`
81	`LDD 204`
82	`CMP 205`
83	`JPN 74`
84	`END`
…	
200	`130`
201	`133`
202	`130`
203	`0`
204	`0`
205	`2`

IX	`0`

ASCII code table (selected codes only)

ASCII code	Character
127	?
128	!
129	“
130	*
131	$
132	&
133	%
134	/

Instruction set

Instruction		Explanation
Op code	Operand	
`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 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).
`DEC`	`<register>`	Subtract 1 from the contents of the register (ACC or IX).
`CMP`	`<address>`	Compare contents of ACC with contents of `<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.
`JMP`	`<address>`	Jump to the given address.
`OUT`		Output to the screen the character whose ASCII value is stored in ACC.
`END`		Return control to the operating system.

Complete the trace table for the given assembly language program.

Instruction address	ACC	Memory address						IX	O
		200	201	202	203	204	205		
70	130	130	133	130	0	0	2	0	

[8]

QUESTION 18.

3 The following table shows assembly language instructions for a processor which purpose register, the Accumulator (ACC) and an Index Register (IX).

Instruction		**Explanation**
Op code	**Operand**	
`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 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).
`DEC`	`<register>`	Subtract 1 from the contents of the register (ACC or IX).
`CMP`	`<address>`	Compare contents of ACC with contents of <address>.
`JPE`	`<address>`	Following compare instruction, jump to <address> if the compare was True.
`JPN`	`<address>`	Following compare instruction, jump to <address> if the compare was False.
`JMP`	`<address>`	Jump to the given address.
`OUT`		Output to the screen the character whose ASCII value is stored in ACC.
`END`		Return control to the operating system.

(a) **(i)** State what is meant by **absolute addressing** and **symbolic addressing**.

Absolute addressing ....

....

Symbolic addressing ....

....

[2]

(ii) Give an example of an `ADD` instruction using both absolute addressing and symbolic addressing.

Absolute addressing ....

Symbolic addressing ....

[2]

(b) **(i)** State what is meant by **indexed addressing** and **immediate addressi**

Indexed addressing ....

....

Immediate addressing ....

.... [2]

(ii) Give an example of an instruction that uses:

Indexed addressing ....

Immediate addressing .... [2]

(c) The current contents of a general purpose register (X) are:

X	1	1	0	0	0	0	0	1

(i) The contents of X represent an unsigned binary integer.

Convert the value in X into denary.

.... [1]

(ii) The contents of X represent an unsigned binary integer.

Convert the value in X into hexadecimal.

.... [1]

(iii) The contents of X represent a two's complement binary integer.

Convert the value in X into denary.

.... [1]

(d) The current contents of the main memory, Index Register (IX) and selected ASCII character set are:

Address	Instruction
40	`LDD 100`
41	`CMP 104`
42	`JPE 54`
43	`LDX 100`
44	`CMP 105`
45	`JPN 47`
46	`OUT`
47	`LDD 100`
48	`DEC ACC`
49	`STO 100`
50	`INC IX`
51	`JMP 41`
52	
53	
54	`END`
…	
100	`2`
101	`302`
102	`303`
103	`303`
104	`0`
105	`303`

IX	`1`

ASCII code table (selected codes only)

ASCII code	Character
300	/
301	*
302	-
303	+
304	^
305	=

This is a copy of the instruction set.

Instruction		Explanation
Op code	Operand	
`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 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).
`DEC`	`<register>`	Subtract 1 from the contents of the register (ACC or IX).
`CMP`	`<address>`	Compare contents of ACC with contents of <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.
`JMP`	`<address>`	Jump to the given address.
`OUT`		Output to the screen the character whose ASCII value is stored in ACC.
`END`		Return control to the operating system.

Complete the trace table for the given assembly language program.

Instruction address	ACC	Memory address						IX	OUT
		100	101	102	103	104	105		
		2	302	303	303	0	303	1	
40									

[7]

QUESTION 19.

2 The following table shows assembly language instructions for a processor which purpose register, the Accumulator (ACC) and an Index Register (IX).

Instruction		Explanation
Op code	**Operand**	
`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 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).
`DEC`	`<register>`	Subtract 1 from the contents of the register (ACC or IX).
`CMP`	`<address>`	Compare contents of ACC with contents of <address>.
`JPE`	`<address>`	Following compare instruction, jump to <address> if the compare was True.
`JPN`	`<address>`	Following compare instruction, jump to <address> if the compare was False.
`JMP`	`<address>`	Jump to the given address.
`OUT`		Output to the screen the character whose ASCII value is stored in ACC.
`END`		Return control to the operating system.

(a) State what is meant by **relative addressing** and **indexed addressing**.

Relative addressing ....

....

....

Indexed addressing ....

....

....

[2]

(b) The current contents of a general purpose register (X) are:

X	1	1	1	1	0	0	1	0

(i) The contents of X represent an unsigned binary integer.

Convert the value in X into denary.

....[1]

(ii) The contents of X represent an unsigned binary integer.

Convert the value in X into hexadecimal.

....[1]

(iii) The contents of X represent a two's complement binary integer.

Convert the value in X into denary.

....[1]

(iv) Show the result on the general purpose register (X) after the following instruction is run.

```
INC X
```


[1]

(c) The current contents of the main memory, Index Register (IX) and selected ASCII character set are provided with a copy of the instruction set.

Address	Instruction
20	`LDD 96`
21	`CMP 97`
22	`JPE 32`
23	`LDX 86`
24	`CMP 98`
25	`JPN 27`
26	`OUT`
27	`LDD 96`
28	`INC ACC`
29	`STO 96`
30	`INC IX`
31	`JMP 21`
32	`END`
…	
93	`453`
94	`453`
95	`452`
96	`8`
97	`10`
98	`453`

IX	`8`

ASCII code table (selected codes only)

ASCII code	Character
450	<
451	>
452	=
453	&
454	(
455	)

Instruction set

Instruction		Explanation
Op code	**Operand**	
`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 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).
`DEC`	`<register>`	Subtract 1 from the contents of the register (ACC or IX).
`CMP`	`<address>`	Compare contents of ACC with contents of <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.
`JMP`	`<address>`	Jump to the given address.
`OUT`		Output to the screen the character whose ASCII value is stored in ACC.
`END`		Return control to the operating system.

Complete the trace table for the given assembly language program.

Instruction address	ACC	Memory address						IX	O
		93	94	95	96	97	98		
		453	453	452	8	10	453	8	
20									

[7]

QUESTION 20.

5 A simple program written in assembly language is translated using a two-pass as

(a) The table contains some of the tasks performed by a two-pass assembler.

Tick (✓) **one** box in each row to indicate whether the task is performed at the first or se pass. The first row has been completed for you.

Task	First pass	Second pass
Creation of symbol table	✓	
Expansion of macros		
Generation of object code		
Removal of comments		

[2]

(b) The processor's instruction set can be grouped according to their function. For example, one group is modes of addressing.

Identify **two** other groups of instructions.

1 ....

....

2 ....

....

[2]

(c) The table shows assembly language instructions for a processor which purpose register, the Accumulator (ACC), and an Index Register (IX).

Instruction		Explanation
Op code	**Operand**	
`LDM`	`#n`	Immediate addressing. Load the denary 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 denary number n to IX.
`STO`	`<address>`	Store 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 contents of ACC with denary 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.
`JMP`	`<address>`	Jump to the given address.
`OUT`		Output to screen the character whose ASCII value is stored in ACC.
`END`		Return control to the operating system.

The current contents of the main memory, Index Register (IX) and selected values from the ASCII character set are:

Address	Instruction
20	`LDM #0`
21	`STO 300`
22	`CMP #0`
23	`JPE 28`
24	`LDX 100`
25	`ADD 301`
26	`OUT`
27	`JMP 30`
28	`LDX 100`
29	`OUT`
30	`LDD 300`
31	`INC ACC`
32	`STO 300`
33	`INC IX`
34	`CMP #2`
35	`JPN 22`
36	`END`
…	
100	`65`
101	`67`
102	`69`
103	`69`
104	`68`
…	
300	
301	`33`
IX	`0`

ASCII code table (Selected codes only)

ASCII Code	Character
65	A
66	B
67	C
68	D
69	E
97	a
98	b
99	c
100	d
101	e

Trace the program currently in memory using the following trace table. Th
has been completed for you.

Instruction address	ACC	Memory address							IX	OU
		100	101	102	103	104	300	301		
		65	67	69	69	68		33	0	
20	0									

[8]

QUESTION 21.

3 The fetch-execute cycle is shown in register transfer notation.

```
01      MAR ← [PC]

02      PC ← [PC] - 1

03      MDR ← [MAR]

04      CIR ← [MAR]
```

(a) There are **three** errors in the fetch-execute cycle shown.

Identify the line number of each error and give the correction.

Line number ....

Correction ....

Line number ....

Correction ....

Line number ....

Correction ....
[3]

(b) A processor's instruction set can be grouped according to their function. For example, one group is the input and output of data.

Identify **two** other groups of instructions.

1 ....

....

2 ....

....
[2]

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

Instruction		Explanation
Op code	**Operand**	
`LDM`	`#n`	Immediate addressing. Load the denary 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 denary number n to IX
`STO`	`<address>`	Store 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 contents of ACC with denary 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
`JMP`	`<address>`	Jump to the given address
`OUT`		Output to the screen the character whose ASCII value is stored in ACC
`END`		Return control to the operating system

The current contents of the main memory, Index Register (IX) and selected values from the ASCII character set are:

Address	Instruction
50	`LDM #0`
51	`STO 401`
52	`LDX 300`
53	`CMP #0`
54	`JPE 62`
55	`ADD 400`
56	`OUT`
57	`LDD 401`
58	`INC ACC`
59	`STO 401`
60	`INC IX`
61	`JMP 52`
62	`END`
…	
300	2
301	5
302	0
303	4
…	
400	64
401	

IX	0

ASCII code table (Selected codes only)

ASCII code	Character
65	A
66	B
67	C
68	D
69	E

Trace the program currently in memory using the following trace table.
The first instruction has been completed for you.

Instruction address	ACC	Memory address						IX	OUTPUT
		300	301	302	303	400	401		
		2	5	0	4	64		0	
50	0								

[8]

(d) The ASCII character code for ‘A’ is 65 in denary.

(i) Convert the denary ASCII character code for ‘A’ into 8-bit binary.

[1]

(ii) Convert the denary ASCII character code for ‘A’ into hexadecimal.

.... [1]

(iii) The Unicode character code for ‘G’ is 0047 in hexadecimal.

State, in hexadecimal, the Unicode character code for ‘D’.

.... [1]

QUESTION 22.

4 A program is written in assembly language.

(a) The op codes `LDM` and `LDD` are used to load a register. The op code `LDM` uses addressing, and the op code `LDD` uses direct addressing.

Describe what happens when the following instructions are run.

```
LDM #300
```

....

....

```
LDD 300
```

....

....

[2]

(b) Assembly language instructions can be grouped by their purpose.

The following table shows four assembly language instructions.

Tick (✓) **one** box in each row to indicate the group each instruction belongs to.

Instruction	Description	Jump instruction	Arithmetic operation	Data movement
`LDR #3`	Load the number 3 to the Index Register			
`ADD #2`	Add 2 to the Accumulator			
`JPN 22`	Move to the instruction at address 22			
`DEC ACC`	Subtract 1 from the Accumulator			

[3]

(c) The processor handles interrupts within the fetch-execute cycle.

(i) Give **one** example of a hardware interrupt and **one** example of a software int

Hardware ....

....

Software ....

....

[2]

(ii) Explain how the processor handles an interrupt.

....

....

....

....

....

....

....

....

....

.... [5]