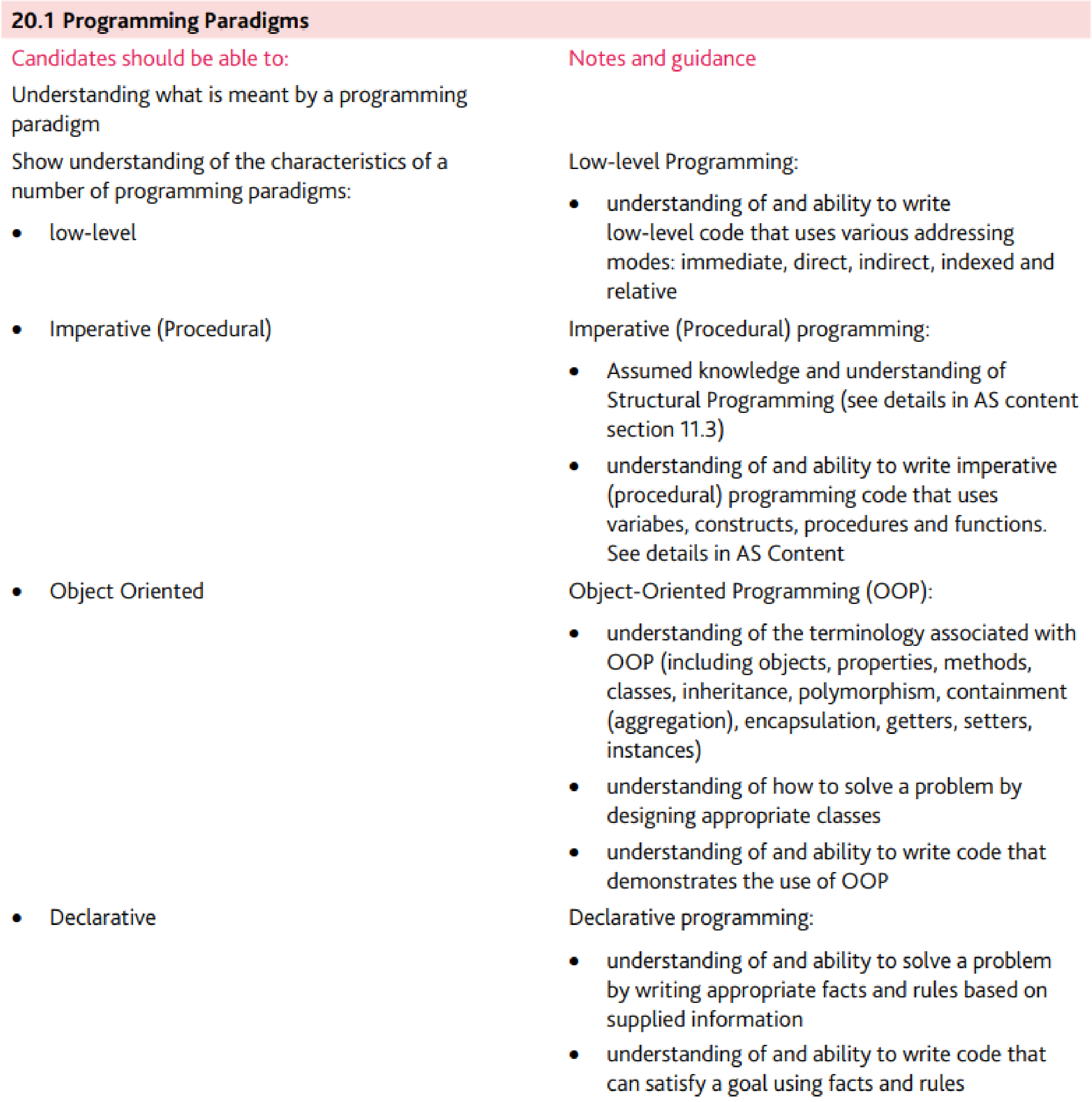
1. Low-level



20.1 Programming Paradigms	
Candidates should be able to:	Notes and guidance
Understanding what is meant by a programming paradigm	
Show understanding of the characteristics of a number of programming paradigms:	Low-level Programming:
• low-level	• understanding of and ability to write low-level code that uses various addressing modes: immediate, direct, indirect, indexed and relative
• Imperative (Procedural)	Imperative (Procedural) programming: • Assumed knowledge and understanding of Structural Programming (see details in AS content section 11.3) • understanding of and ability to write imperative (procedural) programming code that uses variabes, constructs, procedures and functions. See details in AS Content
• Object Oriented	Object-Oriented Programming (OOP): • understanding of the terminology associated with OOP (including objects, properties, methods, classes, inheritance, polymorphism, containment (aggregation), encapsulation, getters, setters, instances) • understanding of how to solve a problem by designing appropriate classes • understanding of and ability to write code that demonstrates the use of OOP
• Declarative	Declarative programming: • understanding of and ability to solve a problem by writing appropriate facts and rules based on supplied information • understanding of and ability to write code that can satisfy a goal using facts and rules

---

> **Programming paradigms**: A programming style/classification // characteristics/features that programming language has/uses

Low-level

Modes of addressing	**Definition**
Immediate	The operand is the data being used
Direct	The operand is the address of the data being used
Indirect	The content stored in the content of the operand is the data being used
Relative	The address of data being used is the current address add to the operand
Indexed	The address of the data being used is the content of the operand added to the content of index register

Instruction		Explanation
Opcode	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
MOV	<register>	Move the contents of the accumulator to the given register (IX)
STO	<address>	Store the contents of ACC at the given address
ADD	<address>	Add the contents of the given address to the ACC
ADD	#n	Add the denary number n to the ACC
SUB	<address>	Subtract the contents of the given address from the ACC
SUB	#n	Subtract the denary number n from 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
CMI	<address>	Indirect addressing. The address to be used is at the given address. Compare the contents of ACC with the contents of this second 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
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

All questions will assume there is only one general purpose register available (Accumulator)

ACC denotes Accumulator
IX denotes Index Register

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