

20.1.1 The five addressing modes, and declarative programming with facts, rules and goals

Name: _____ Class: _____ Date: _____ **Total: 29 marks**

KEY WORDS object-oriented programming 面向对象编程 • methods 方法 • rule 规则
• declarative programming 声明式编程 • goal 目标

Objective

Build the skills to answer the **programming paradigm** questions that sheet 20.1 does not reach. That sheet is about object-oriented programming: the four pillars, encapsulation, a class diagram. Two of the four paradigms the syllabus names are barely there – the words addressing mode, immediate, indexed and relative occur **zero** times in it, and **goal** does too. This sheet is **low-level** programming through its five **addressing modes** 寻址方式, and **declarative** programming through **facts**, **rules** and a **goal**.

You must be able to:

- write and trace low-level code that uses the **immediate**, **direct**, **indirect**, **indexed** and **relative** addressing modes
- solve a problem by writing appropriate **facts** 事实 and **rules** 规则 from supplied information
- write code that can satisfy a **goal** 目标 using facts and rules
- identify the paradigm a code sample belongs to, and describe the characteristics of each

1 Worked examples

Study these first. Each one shows a **method** that a question later on this sheet needs.

■ The five addressing modes

address	contents	mode	instruction	ACC receives
105	27	immediate	LDM #105	105 (the operand itself)
106	64	direct	LDD 105	27 (contents of 105)
107	105	indirect	LDI 105	91 (105 holds 27, so contents of 27)
27	91	indexed	LDX 105	105 (contents of 105 + IX = 107)
145	16	relative	JMR +65	next address 40 + 65 = 105

IX = 2, instruction at address 40

Every one of these instructions writes the **same operand**, 105. What differs is how many times the processor follows it.

Take this memory, with the index register IX holding 2:

Address	27	105	106	107
Contents	91	27	64	43

Instruction	Mode	What ACC becomes	Why
LDM #105	immediate	105	the operand is the value; # marks it
LDD 105	direct	27	the operand is the address of the value
LDI 105	indirect	91	105 holds 27, and 27 is the address actually used
LDX 105	indexed	43	the address is 105 + IX = 107
JMR +6	relative	–	jump to an offset from the current instruction, not to a fixed address

- Count the steps: immediate follows the operand **zero** times, direct **once**, indirect **twice**. That is the whole distinction, and it is what a trace question tests.
- **Indexed** exists for **arrays**: put the array's start address in the instruction and the subscript in IX, then increment IX to step along it, with the instruction itself unchanged.
- **Relative** exists for **relocatable** code: the program works wherever it is loaded, because every jump is measured from where it already is rather than from address zero.

■ Declarative programming: facts, rules and goals

In a **declarative** language the program states **what is known** and **what is wanted**. The programmer writes no sequence of steps; the language's **inference engine** 推理引擎 searches for a way to satisfy the goal.

- A **fact** states something that is true, with no condition: `book(dracula, stoker)`.
- A **rule** states something that is true **provided** other things are: `classic(X) IF published(X, Y) AND Y < 1900`.

- A **goal** is the question asked of the program: `classic(dracula)`.

Worked example. *A library holds these facts.*

```
book(dracula, stoker).
book frankenstein, shelley.
published(dracula, 1897).
published frankenstein, 1818).
```

Write a rule that defines a *classic* as a book published before 1900, and a goal that lists every *classic*.

```
classic(X) IF published(X, Y) AND Y < 1900
```

```
classic(W)
```

To satisfy `classic(W)`, the engine takes each fact matching `published(W, Y)` in turn, binds `Y`, and tests `Y < 1900`. Both succeed, so it returns `dracula` and `frankenstein`.

- A **variable** starts with a capital letter and a **constant** with a small one. Writing `classic(Dracula)` asks a different question – it is a variable, so it matches anything.
- The order of the facts does not change the answer. That is the point of the paradigm: there is no flow of control to get wrong.
- A goal with a **constant** asks “is this true?” and answers yes or no; a goal with a **variable** asks “which values make this true?” and returns them.

■ Naming the paradigm from a sample

Sample	Paradigm	The giveaway
LDD 200, ADD #5, STO 201	low-level	mnemonics, registers, memory addresses
FOR Count <- 1 TO 10 ... NEXT Count	imperative	a sequence of statements that change state
CLASS Dog ... PRIVATE Name : STRING	object-oriented	data and methods bound together in a class
type(lion, wild). and dangerous(X) IF type(X, wild)	declarative	facts and rules, with no order of execution

2 Practice

Now use the methods above.

Questions 2.1 and 2.2 use this memory, with `IX` holding 2:

Address	27	105	106	107
Contents	91	27	64	43

2.1 Give the contents of `ACC` after each instruction is executed on its own. [4]

- LDM #105
- LDD 105
- LDI 105
- LDX 105

2.2 Name the addressing mode used by each instruction in 2.1. [2]

2.3 State the difference between a **fact** and a **rule** in a declarative language. [2]

2.4 Name the paradigm each sample belongs to. [3]

(a) LDD 200 then ADD #5 then STO 201

(b) CLASS Dog with PRIVATE Name : STRING

(c) type(lion, wild). and dangerous(X) IF type(X, wild)

3 Exam-style questions

3.1 Explain why **indexed** addressing is used to process an array, and why **relative** addressing makes a program **relocatable**. [4]

3.2 State what is meant by **indirect** addressing, and give **one** situation in which it is needed. [3]

3.3 A vet's surgery records its patients. Each animal has an owner and a species.

(a) Write **facts** to record that Bella is a dog owned by Chen, and that Milo is a cat owned by Rani. [4]

(b) Write a **rule** that defines a **dogowner** as someone who owns an animal whose species is dog. [2]

(c) Write the **goal** that would list every dog owner, and state what the inference engine returns. [2]

3.4 Explain how a declarative program differs from an imperative one, in terms of what the programmer writes. [3]

Solutions

Each answer shows the steps, then the **result**.

2.1

- (a) LDM #105: the operand is the value, so ACC = **105**
- (b) LDD 105: the contents of address 105, so ACC = **27**
- (c) LDI 105: 105 holds 27, so the contents of address 27, ACC = **91**
- (d) LDX 105: the address is $105 + 2 = 107$, so ACC = **43**

2.2

- (a) **immediate**, (b) **direct**
- (c) **indirect**, (d) **indexed**

2.3

- a **fact** states something that is unconditionally true
- a **rule** states something that is true **only if** one or more other conditions are satisfied

2.4

- (a) **low-level**
- (b) **object-oriented**
- (c) **declarative**

3.1

- in **indexed** addressing the effective address is the operand plus the index register, so the instruction holds the array's **start** address and IX holds the subscript
- incrementing IX moves to the next element with the **instruction itself unchanged**, so one instruction inside a loop can read the whole array
- in **relative** addressing the target is an **offset from the current instruction** rather than an absolute address
- so wherever the program is loaded in memory, every jump still lands in the right place and nothing has to be recalculated: the code is relocatable

3.2

- the operand is the **address of an address**: the processor reads the contents of the given address and uses **that** as the address of the value it wants
- so it follows the operand **twice** rather than once
- it is needed for **pointers**, and wherever the location of the data is not known until run time – for example following a linked list, or reaching a value through a table of addresses

3.3

- (a) `animal(bella, dog).` and `owns(chen, bella).`

- `animal(milo, cat).` and `owns(rani, milo).`
- names are constants and so start with a **small** letter; any consistent predicate names are accepted, provided the same ones are used in (b)

(b) `dogowner(X) IF owns(X, Y) AND animal(Y, dog)`

- one mark for the two conditions, one for joining them with the shared variable Y, which is what links the owner to their animal

(c) `dogowner(W)`

- the engine returns **chen**, the only owner whose animal has species dog

3.4

- in an **imperative** program the programmer writes a **sequence of instructions** that are executed in order and change the program's state: they say **how** the task is done
- in a **declarative** program the programmer writes **facts and rules** – what is known and what is wanted – and no order of execution at all
- the language's **inference engine** then works out how to satisfy the goal, so the programmer says **what** is required rather than how to get it