ANSWERS

A9.4 Instruction sets and addressing modes

Computer architecture · A level · OCR H446 1.2.4, AQA 7517 4.7.3.3, Eduqas A500QS 2.1 · about 20 min

What this lesson is about

Opcodes, operands and instruction formats; immediate, direct, indirect and indexed addressing.

- The instruction set
- Opcode and operand
- Addressing modes
- Indexed addressing walks an array

Questions

6 marks in all

1. Address 30 holds 45 and address 45 holds 8. The operand is 30. What value does indirect addressing use? [1 mark]

- ☐ **A** 8
- ☐ **B** 45
- ☐ **C** 30
- ☐ **D** 75

> **Answer:** A. Indirect: address 30 holds the address (45), and address 45 holds the value, 8.

2. Which addressing mode is best suited to stepping through an array in a loop? [1 mark]

- ☐ **A** Indexed
- ☐ **B** Immediate
- ☐ **C** Direct
- ☐ **D** Indirect

> **Answer:** A. The operand gives the start of the array and the index register is added to it, then incremented each time.

3. In immediate addressing, the operand is... [1 mark]

- ☐ **A** the actual value to be used
- ☐ **B** the address of the value
- ☐ **C** the address of the address of the value
- ☐ **D** added to the index register

> **Answer:** A. No memory access is needed, which makes immediate addressing fast.

4. An instruction is 16 bits: 5 bits of opcode and 11 bits of operand. What is the largest address direct addressing can reach? [1 mark]

> **Answer:** 2047. 11 bits give addresses 0 to 2 to the power 11 minus 1, which is 2047.

5. Why will a program compiled for an x86 processor not run on an ARM processor? [1 mark]

- ☐ **A** The instruction set is processor specific, so the machine code means something different
- ☐ **B** ARM processors have no memory
- ☐ **C** x86 programs are written in assembly
- ☐ **D** ARM processors only run Python

> **Answer:** A. Each processor family has its own opcodes and instruction formats.

6. An 8-bit instruction has a 4-bit opcode then a 4-bit operand. What does this print? [1 mark]

```
instruction = 0b10110110
print(instruction >> 4, instruction & 0b1111)
```

> **Answer:**
>
> ```
> 11 6
> ```
>
> The top four bits 1011 are 11 and the bottom four bits 0110 are 6.

The task: four ways to read an operand

Memory is the list `memory` (addresses 0 to 9) and the index register `ix` holds 2. Write `operand_value(mode, operand)`: - `mode` is one of the strings `"immediate"`, `"direct"`, `"indirect"` or `"indexed"`; - `operand` is a whole number from 0 to 9; - it returns the value an instruction would use in that mode, as defined in the table above. For each operand in `[2, 5]`, and for each mode in the order immediate, direct, indirect, indexed, print a line in the form `direct 2 -> 1`: the mode, the operand, `->`, then the value. That is eight lines. The robot does not move.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

memory = [4, 7, 1, 9, 5, 3, 8, 0, 6, 2]
ix = 2

def operand_value(mode, operand):
    return operand
```

The hint students can ask for: For each mode, ask how many times memory is looked up and with which address. Immediate needs no look-up, direct one, indirect uses the result of one look-up as the address for another, and indexed adjusts the address using the index register before looking up.

A solution

```
from bugbot import *
connect()
memory = [4, 7, 1, 9, 5, 3, 8, 0, 6, 2]
ix = 2

def operand_value(mode, operand):
    if mode == "immediate":
        return operand
    if mode == "direct":
        return memory[operand]
    if mode == "indirect":
        return memory[memory[operand]]
    if mode == "indexed":
        return memory[operand + ix]

for operand in [2, 5]:
    for mode in ["immediate", "direct", "indirect", "indexed"]:
        print(mode, operand, "->", operand_value(mode, operand))
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.