ANSWERS

A9.10 Project: a processor of your own

Computer architecture · A level · OCR H446 1.1.1, AQA 7517 4.7.1.1, Eduqas A500QS 2.1 · about 35 min

What this lesson is about

Build a von Neumann machine with register transfers, addressing modes, flags and memory-mapped I/O that plays a scale on the robot.

- The machine
- The program in memory
- Step 1: fetch
- Step 2: decode and the addressing modes
- Step 3: store and memory-mapped I/O
- Step 4: flags and branches
- Step 5: finish

Questions

4 marks in all

1. In the project machine, storing a value at address 255 plays a note. What is this technique called? [1 mark]

- ☐ **A** Memory-mapped I/O
- ☐ **B** Indirect addressing
- ☐ **C** Pipelining
- ☐ **D** Virtual storage

> **Answer:** A. The address on the bus selects an I/O controller instead of a memory location.

2. After CMP #700 with 650 in the accumulator, which flags are set in the project machine? [1 mark]

- ☐ **A** N = 1 and Z = 0
- ☐ **B** N = 0 and Z = 1
- ☐ **C** N = 0 and Z = 0
- ☐ **D** N = 1 and Z = 1

> **Answer:** A. 650 - 700 is -50: negative, not zero, so BLT branches.

3. The loop in the project program is 6 instructions long and runs 4 times. With 4 instructions before it and HLT after it, how many fetches are there? [1 mark]

> **Answer:** 29. 4 + 6 × 4 + 1 = 29.

4. What does this program print? [1 mark]

```
memory = ['LDA #5', 'ADD 5', 'STA 6', 'HLT', 0, 7, 0]
pc = acc = 0
while True:
    cir = memory[pc]
    pc += 1
    op, _, x = cir.partition(' ')
    if op == 'HLT':
        break
    value = int(x[1:]) if x.startswith('#') else memory[int(x)]
    if op == 'LDA': acc = value
    elif op == 'ADD': acc += value
    elif op == 'STA': memory[int(x)] = acc
print(memory[6], pc)
```

Answer:

```
12 4
```

LDA #5 is immediate, ADD 5 adds the contents of address 5 (7), and the total 12 is stored at address 6; HLT was fetched from address 3, so the PC is 4.

The task: build the processor

Build the machine described above and run the program. The starter loads the program into memory and sets up the registers; you write everything else, using exactly the variable names `pc`, `mar`, `mdr`, `cir` and `acc` for the registers. - After every fetch, print `PC=<pc> MAR=<mar> CIR=<cir>` with the register values straight after the four fetch transfers, for example `PC=10 MAR=9 CIR=BLT 4`. - Storing to address 254 calls `forward(50, distance=<value>)`; storing to address 255 calls `tone(<value>, 0.2)`. - When `HLT` is fetched, stop and print `halted after <cycles> cycles`, counting every fetch including the `HLT`, then `ACC=<acc> Z=<z> N=<n>`. A working machine drives 10 cm, plays 300, 400, 500 and 600 Hz, prints 29 fetch lines, then `halted after 29 cycles` and `ACC=700 Z=1 N=0`.

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

memory = [0] * 256
program = [
    "LDA #10", "STA 254", "LDA #300", "STA 20", "LDA 20", "STA 255",
    "ADD #100", "STA 20", "CMP #700", "BLT 4", "HLT",
]
memory[:len(program)] = program

pc, mar, mdr, cir, acc = 0, 0, 0, "", 0
z, n = 0, 0
cycles = 0
```

The hint students can ask for: Build it in the order of the steps and test after each one. Write one helper that turns an operand into a value, so immediate and direct addressing are handled in one place. Check the address before a store, so 254 and 255 reach the robot instead of memory, and let the branches look only at the flags.

A solution

```
from bugbot import *
connect()
memory = [0] * 256
program = [
    "LDA #10", "STA 254", "LDA #300", "STA 20", "LDA 20", "STA 255",
    "ADD #100", "STA 20", "CMP #700", "BLT 4", "HLT",
]
memory[:len(program)] = program

pc, mar, mdr, cir, acc = 0, 0, 0, "", 0
z, n = 0, 0
cycles = 0

def value(operand):
    global mar, mdr
    if operand.startswith("#"):
        return int(operand[1:])
    mar = int(operand)
    mdr = memory[mar]
    return mdr

while True:
    mar = pc
    pc = pc + 1
```

```
    mdr = memory[mar]
    cir = mdr
    cycles = cycles + 1
    print(f"PC={pc} MAR={mar} CIR={cir}")
    opcode, _, operand = cir.partition(" ")
    if opcode == "HLT":
        break
    elif opcode == "LDA":
        acc = value(operand)
    elif opcode == "ADD":
        acc = acc + value(operand)
    elif opcode == "SUB":
        acc = acc - value(operand)
    elif opcode == "STA":
        mar = int(operand)
        mdr = acc
        if mar == 254:
            forward(50, distance=mdr)
        elif mar == 255:
            tone(mdr, 0.2)
        else:
            memory[mar] = mdr
    elif opcode == "CMP":
        result = acc - value(operand)
        z = 1 if result == 0 else 0
        n = 1 if result < 0 else 0
    elif opcode == "BRA":
        pc = int(operand)
    elif opcode == "BEQ":
        if z == 1:
            pc = int(operand)
    elif opcode == "BLT":
        if n == 1:
            pc = int(operand)

print(f"halted after {cycles} cycles")
print(f"ACC={acc} Z={z} N={n}")
```

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