WORKSHEET

# A9.3 The fetch-decode-execute cycle in detail

Computer architecture · A level · OCR H446 1.1.1, AQA 7517 4.7.3.2, Eduqas A500QS 2.1 · about 25 min

Name _____ Class _____ Date _____

## What this lesson is about

Register transfer notation, the buses at each step, a full register trace, and a stored program that plays notes.

- Register transfer notation
- Fetch
- Decode
- Execute
- A full trace
- Memory-mapped output

## Questions

**5 marks in all**

**1.** Put the fetch stage in order. [1 mark]

Number the lines 1 to 4 to put them in the right order.

___ `MDR ← [Memory]addressed`

___ `CIR ← [MDR]`

___ `PC ← [PC] + 1`

___ `MAR ← [PC]`

**2.** During the fetch, which bus carries the instruction from memory to the MDR? [1 mark]

- ☐ **A** Data bus
- ☐ **B** Address bus
- ☐ **C** Control bus
- ☐ **D** None: it is already in the processor

**3.** What does this program print? [1 mark]

```
memory = ['LOAD 3', 'ADD 4', 'HALT', 15, 27]
pc = 0
acc = 0
while True:
    mar = pc
    pc = pc + 1
    mdr = memory[mar]
    cir = mdr
    op = cir.split()
    if op[0] == 'HALT':
        break
    mar = int(op[1])
    mdr = memory[mar]
    acc = mdr if op[0] == 'LOAD' else acc + mdr
print(pc, mar, acc)
```

---

---

---

**4.** The PC holds 12 at the start of a fetch. What does the PC hold while that instruction is executed, if it is not a branch? [1 mark]

- ☐ **A** 13
- ☐ **B** 12
- ☐ **C** 11
- ☐ **D** It depends on the operand

**5.** How does a branch instruction change the flow of a program? [1 mark]

- ☐ **A** Its execute step loads a new address into the PC
- ☐ **B** It changes the MAR
- ☐ **C** It rewrites the instructions in memory
- ☐ **D** It sends an interrupt

## The task: fetch in register transfers

The program below already decodes and executes. Write the **fetch** as four register transfers, in the order in the table above, using the variables `pc`, `mar`, `mdr` and `cir`. Straight after the fetch, before decoding, print the four registers in the form `PC=1 MAR=0 MDR=LOAD 7 CIR=LOAD 7`. Memory holds `LOAD 7`, `OUT`, `ADD 8`, `OUT`, `ADD 8`, `OUT`, `HALT`, then the data 300 and 100. `OUT` plays the accumulator for 0.2 seconds, so a working fetch plays 300, 400 and 500 Hz and prints seven lines, the last `PC=7 MAR=6 MDR=HALT CIR=HALT`. The robot does not move.

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

memory = ["LOAD 7", "OUT", "ADD 8", "OUT", "ADD 8", "OUT", "HALT", 300, 100]
pc, mar, mdr, cir, acc = 0, 0, 0, "", 0

while True:
    # fetch: four register transfers, then print the registers
    cir = "HALT"

    # decode
    parts = cir.split()
    opcode = parts[0]
    # execute
    if opcode == "LOAD":
        mar = int(parts[1]); mdr = memory[mar]; acc = mdr
    elif opcode == "ADD":
        mar = int(parts[1]); mdr = memory[mar]; acc = acc + mdr
    elif opcode == "OUT":
        tone(acc, 0.2)
    elif opcode == "HALT":
        break
```

**Plan your program here, then type it in and press Run.**

**Do it on the robot**

www.bugbotlab.com/learn/a9-3-fetch-decode-execute-in-detail/

The simulator checks it and tells you when it passes. Nothing to install, no account.

## Challenges

1. Add a `BRANCH n` instruction that sets the PC, and make the program play its three notes twice.
2. Add `STORE n` with memory-mapped output: storing to address 99 plays the value as a note.
3. Write out the register transfers for executing `SUB 9`.