# AQA Computer Science A-Level

# 4.7.3 Structure and role of the processor and its components

Past Paper Questions

# Additional Specimen AS Paper 2

**0 8** There are many system design issues that affect processor performance and the use of multiple cores and cache memory are two of these.

Name and explain **two** more system hardware decisions that will effect processor performance and describe all **four** effects. Use examples to illustrate your description.

**[6 marks]**

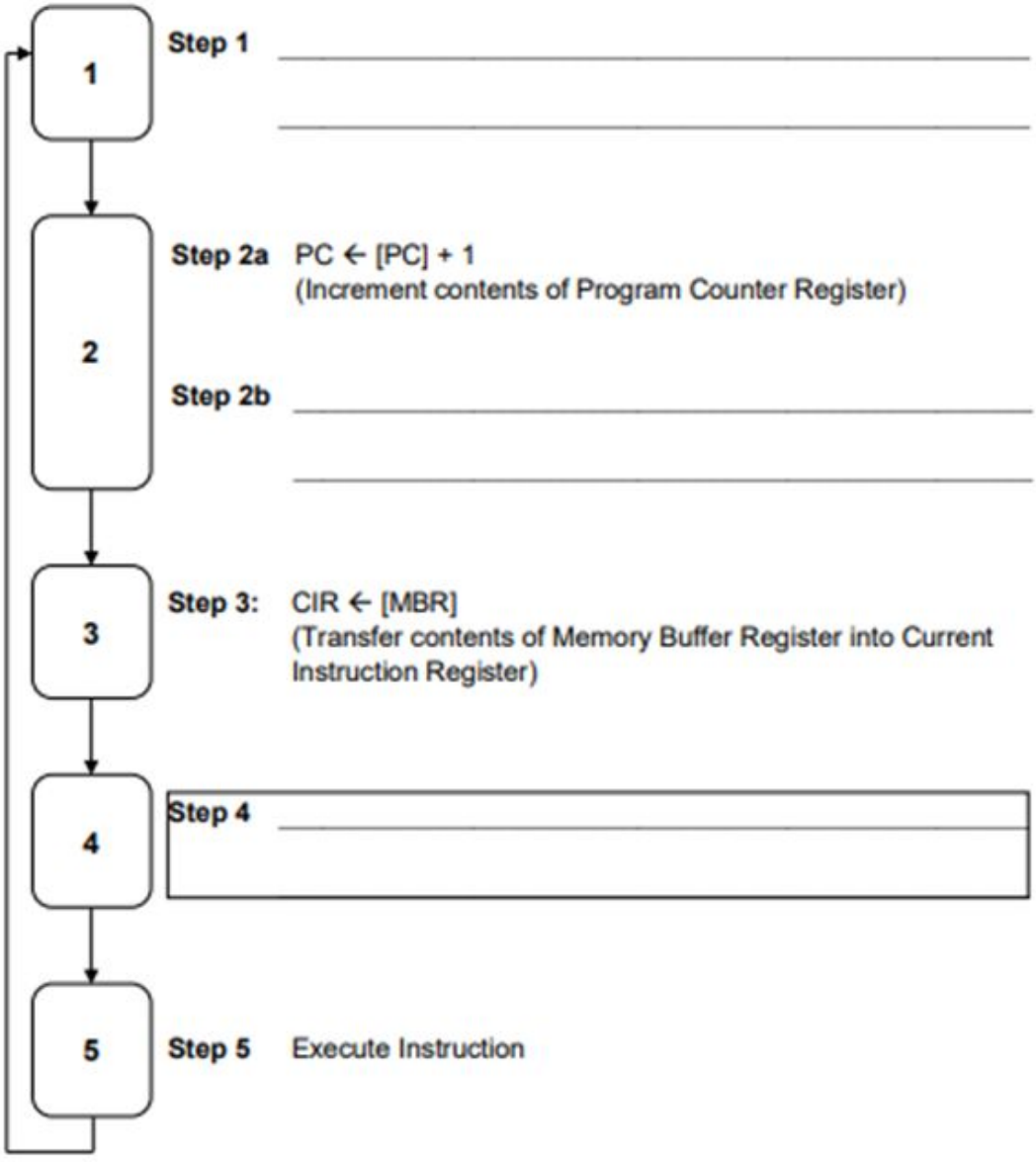
# Additional Specimen Paper 2

0 4 . 3 **Figure 2** shows the fetch-execute cycle, excluding how interrupts are dealt with. Some of the steps in the cycle have been described.

Describe the missing steps 1, 2b and 4 using either register transfer notation or a written description. Steps 2a and 2b occur at the same time.

**[3 marks]**

**Figure 2**



0 4 . 4 An interrupt can occur during the fetch-execute cycle.

Explain what happens when an interrupt occurs.

**[4 marks]**

## January 2009 Comp 2

3 (a) The high-level language statement

```
A := B + 5
```

is to be written in assembly language.

Complete the following assembly language statements, which are to be the equivalent of the above high level language statement. The `Load` and `Store` instructions imply the use of the accumulator register.

```
Load ..........

..........  #5

Store .........
```

*(3 marks)*

**5** The Fetch-Execute cycle can be described in register-transfer language as follows:

MAR ← [PC]
PC ← [PC] + 1; MBR ← $[\text{Memory}]_{\text{addressed}}$
CIR ← [MBR]
[CIR] decoded and executed

where [ ] means *contents of*.

Describe the Fetch-Execute cycle in your own words.

.....

.....

.....

.....

.....

.....

.....

.....

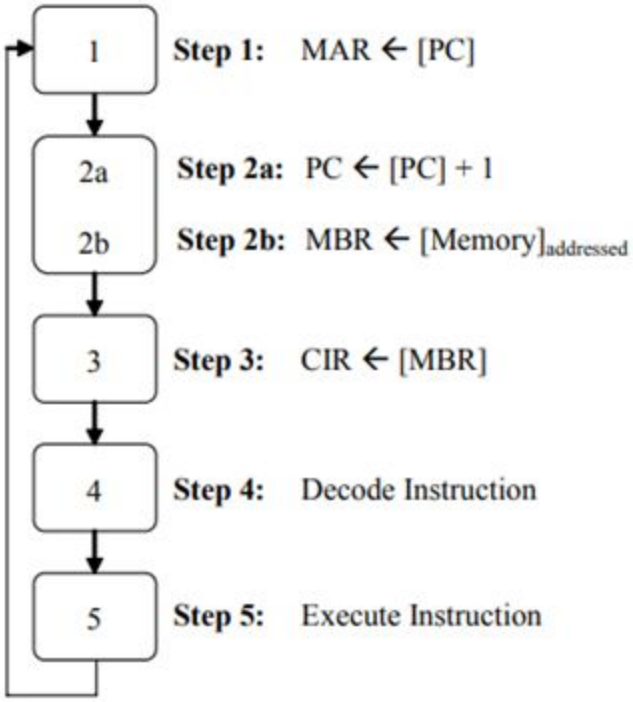
.....

*(6 marks)*

# January 2010 Comp 2

**3** **Figure 1** shows the fetch-execute cycle. Steps 2a and 2b occur at the same time.

**Figure 1**



**3** (a) State the full names of **two** of the special purpose registers that are used in the fetch part of the fetch-execute cycle.

Register 1: .....

Register 2: .....

*(2 marks)*

**3** (b) Explain the role of the address bus, data bus and main memory during Steps 1 and 2b.

.....

.....

.....

.....

.....

*(2 marks)*

**3** (c) Give **one** reason why Steps 2a and 2b are able to occur at the same time.

.....

.....

*(1 mark)*

**5** A single accumulator microprocessor supports the assembly language instructions:

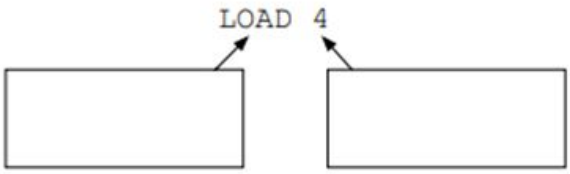
```
LOAD   memory reference
ADD    memory reference
STORE  memory reference
```

An example instruction is:

```
LOAD 4
```

which would copy the contents of the referenced memory location 4 into the accumulator register.

**5** (a) (i) Identify which part of the instruction is the *operand* and which part is the *opcode* by writing the words operand and opcode in the two boxes below.




*(1 mark)*

**5** (a) (ii) The accumulator is a general purpose register.

What is a *register*?

.....

.....

*(1 mark)*

**5** (b) Using the given assembly language instructions, write an assembly language program that adds together the values stored in memory locations 12 and 13, storing the resulting total in memory location 14.

.....

.....

.....

.....

.....

*(3 marks)*

# January 2011 Comp 2

**4** Some of the assembly language instructions supported by a simple microprocessor are:

| Assembly Language |
|---|
| STORE |
| LOAD |
| ADD |

Examples of the use of these assembly language instructions are:

| | |
|---|---|
| STORE 5 | Copy the contents of the accumulator into memory location 5 |
| LOAD 5 | Copy the contents of memory location 5 into the accumulator |
| ADD 2 | Add the contents of memory location 2 to the current contents of the accumulator, storing the result in the accumulator |

**4 (a)** Write into the table below the opcode and the operand parts of the following instruction.

```
LOAD    5
```

| | |
|---|---|
| **Operand** | |
| **Opcode** | |

*(1 mark)*

**4 (b)** Write an assembly language program, using the instructions given above, that adds the contents of memory locations 7, 8 and 3, storing the answer in memory location 21.

.....

.....

.....

.....

.....

.....

.....

*(3 marks)*

**7** A programmer could use either an assembly language or a high level language to code programs for sale.

**7 (a)** Give **two** limitations of using assembly language to code a program.

1 .....

.....

2 .....

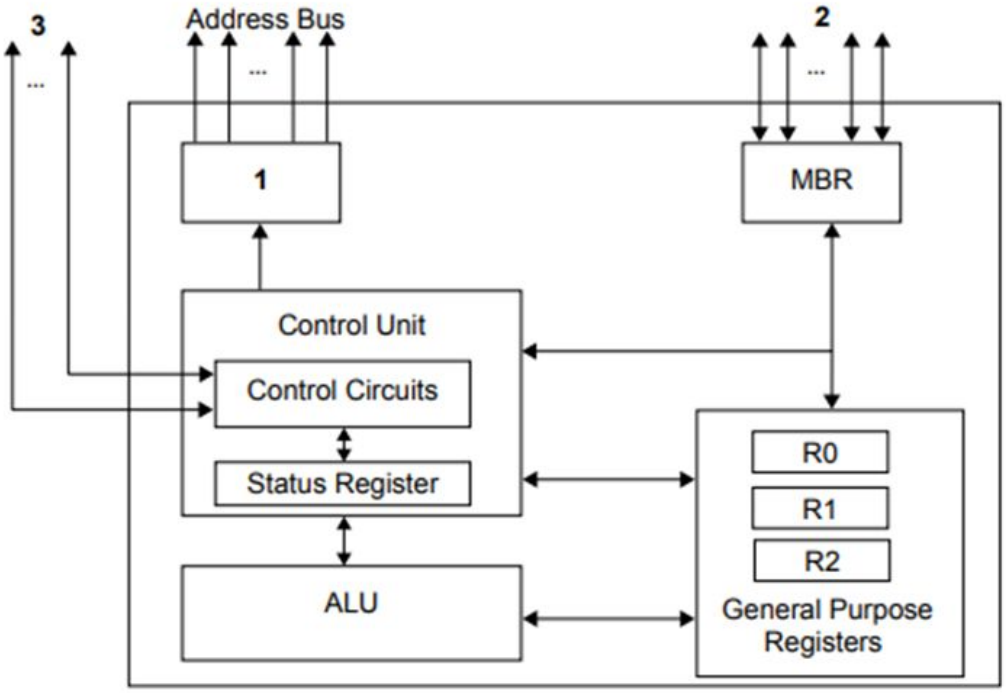
.....

*(2 marks)*

# January 2012 Comp 2

**3** **Figure 4** below shows an incomplete diagram of the components of a processor.

**Figure 4**




**3 (a)** Provide the full names for the components numbered **1** to **3** in **Figure 4** by completing **Table 1** below.

**Table 1**

| Component Number | Component Name |
|---|---|
| 1 | |
| 2 | |
| 3 | |

*(3 marks)*

**3 (b)** What is the role of the Control Unit?

.....

.....

*(1 mark)*

**3 (c)** State the full name of the processor component that would perform subtraction and comparison operations.

.....

.....

*(1 mark)*

**3 (d)** What is meant by the term *register*?

.....

.....

*(1 mark)*

**3 (e)** State **one** example of when the status register might have a bit set.

.....

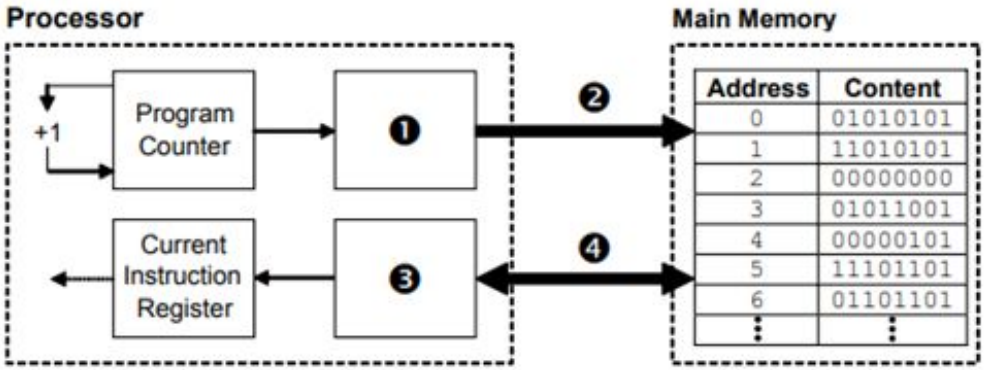
.....

*(1 mark)*

# June 2010 Comp 2

7 **Figure 2** shows the processor registers and busses that are used during the fetch part of the fetch-execute cycle, together with the main memory. The values stored in memory locations 0 to 6 in the main memory are machine code instructions.

**Figure 2**




7 (a) Name the components that are labelled with the numbers ❶ to ❹. In the case of register names, the full names must be stated.

| Number | Component Name |
|---|---|
| ❶ | |
| ❷ | |
| ❸ | |
| ❹ | |

*(4 marks)*

7 (b) Explain what happens during the decode and execute stages of the fetch-execute cycle.

.....

.....

.....

.....

.....

.....

*(3 marks)*

# June 2011 Comp 2

**4** **Figure 2** and **Figure 3** show different versions of the same program.

**Figure 2**

| (x) | (y) | (z) |
|---|---|---|
| 200 | LOAD | 7 |
| 201 | ADD | 3 |
| 202 | ADD | 6 |
| 203 | STORE | 255 |

**Figure 3**

| (x) | (y) | (z) |
|---|---|---|
| 200 | 01010110 | 00000111 |
| 201 | 11010000 | 00000011 |
| 202 | 11010000 | 00000110 |
| 203 | 11110000 | 11111111 |

**4 (a)** What generation of programming language is shown in **Figure 2**?

.....

*(1 mark)*

**4 (b)** In both figures there is a column labelled **(x)**.

What would be a suitable heading for this column?

.....

*(1 mark)*

**4 (c)** In both figures the instruction is split into two parts.

What are the names of the instruction parts in columns **(y)** and **(z)**?

**(y)**.....

**(z)**.....

*(2 marks)*

**4 (d)** What is the relationship between the instructions in **Figure 2** and **Figure 3**?

.....

.....

*(1 mark)*

**7 (b)** In the first step of the Fetch-Execute Cycle the contents of the Program Counter are copied into another register.

State the **full name** of this register.

.....

*(1 mark)*

**7 (c)** An item of data or an instruction fetched into the processor is initially loaded into a register.

State the **full name** of this register.

.....

*(1 mark)*

## June 2012 Comp 2

**2 (a)** A machine code instruction can be split into an opcode part and an operand part.

**2 (a) (i)** What does an opcode represent?

.....

.....

*(1 mark)*

**2 (a) (ii)** What does an operand represent?

.....

.....

*(1 mark)*

**5** The following registers are used in the Fetch-Execute cycle:

- Current Instruction Register (CIR)
- Memory Address Register (MAR)
- Memory Buffer Register (MBR)
- Program Counter (PC)
- Status Register (SR)

Describe, **using full sentences**, the steps involved in the Fetch-Execute cycle, making reference to how the registers above are used. Your description should cover the fetch, decode and execute phases of the cycle. You may use the abbreviations given above for the register names in your response; for example PC for Program Counter.

In your answer you will be assessed on your ability to use good English and to organise your answer clearly in complete sentences, using specialist vocabulary where appropriate.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

*(6 marks)*

# June 2013 Comp 2

**2 (a)** State the full names of **two** of the special purpose registers that are used in the fetch part of the fetch-execute cycle.

Register 1 .....

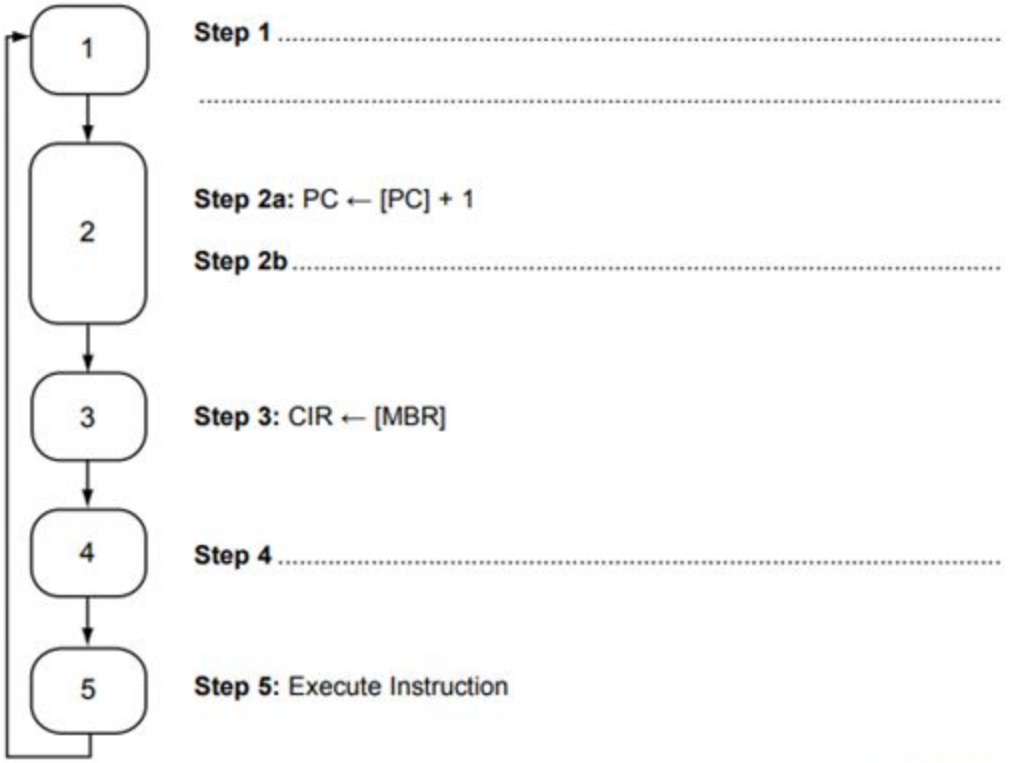
Register 2 .....

*(2 marks)*

**2 (b)** **Figure 1** below is an incomplete diagram of the fetch-execute cycle.

Describe the missing steps 1, 2b and 4 using either register transfer notation or a written description. Steps 2a and 2b occur at the same time.

**Figure 1**




*(3 marks)*

**4** A school robotics club has recently purchased a robotics kit after the teacher in charge saw an advert in a magazine. The advert is reproduced below.

## RoboEddy - a new educational robot

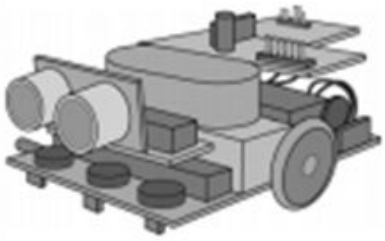
**Specification**

***Hardware***

500 Mhz processor
32 MB RAM
4 timers
Wi-Fi communications via WLAN 802.11g radio
Dual H-bridge motor driver

***Software***

Built in interpreter for the high level language RobotC
Directly run assembly code
XMODEM protocol for reliable file transfer
Support for various analogue and digital sensors



***Available NOW!***

***Only £199***

**4 (d)** As well as using RobotC, it is also possible to program the robot using assembly language.

The motor driver uses memory locations to store the current speed of each motor. The left motor speed is stored in memory location `21` and the right motor speed is stored in memory location `22`.

The following set of three assembly language instructions can be used to take basic control of the motors:

```
LOAD  XX    - load a value from memory location XX into the accumulator
ADD   XX    - add the value stored in memory location XX to the accumulator
STORE XX    - store the value in the accumulator in memory location XX
```

Selecting from the set of three instructions above, write a sequence of instructions that will swap the current left motor speed with the current right motor speed. Your program may use memory location `23` for temporary storage.

.....

.....

.....

.....

.....

.....

*(3 marks)*

**Table 2 – standard AQA assembly language instruction set**

| | |
|---|---|
| `LDR Rd, <memory ref>` | Load the value stored in the memory location specified by `<memory ref>` into register `d`. |
| `STR Rd, <memory ref>` | Store the value that is in register `d` into the memory location specified by `<memory ref>`. |
| `ADD Rd, Rn, <operand2>` | Add the value specified in `<operand2>` to the value in register `n` and store the result in register `d`. |
| `SUB Rd, Rn, <operand2>` | Subtract the value specified by `<operand2>` from the value in register `n` and store the result in register `d`. |
| `MOV Rd, <operand2>` | Copy the value specified by `<operand2>` into register `d`. |
| `CMP Rn, <operand2>` | Compare the value stored in register `n` with the value specified by `<operand2>`. |
| `B <label>` | Always branch to the instruction at position `<label>` in the program. |
| `B<condition> <label>` | Branch to the instruction at position `<label>` if the last comparison met the criterion specified by `<condition>`. Possible values for `<condition>` and their meanings are: `EQ`: equal to `NE`: not equal to `GT`: greater than `LT`: less than |
| `AND Rd, Rn, <operand2>` | Perform a bitwise logical AND operation between the value in register `n` and the value specified by `<operand2>` and store the result in register `d`. |
| `ORR Rd, Rn, <operand2>` | Perform a bitwise logical OR operation between the value in register `n` and the value specified by `<operand2>` and store the result in register `d`. |
| `EOR Rd, Rn, <operand2>` | Perform a bitwise logical XOR (exclusive or) operation between the value in register `n` and the value specified by `<operand2>` and store the result in register `d`. |
| `MVN Rd, <operand2>` | Perform a bitwise logical NOT operation on the value specified by `<operand2>` and store the result in register `d`. |
| `LSL Rd, Rn, <operand2>` | Logically shift left the value stored in register `n` by the number of bits specified by `<operand2>` and store the result in register `d`. |
| `LSR Rd, Rn, <operand2>` | Logically shift right the value stored in register `n` by the number of bits specified by `<operand2>` and store the result in register `d`. |
| `HALT` | Stops the execution of the program. |

**Labels**: A label is placed in the code by writing an identifier followed by a colon (:). To refer to a label, the identifier of the label is placed after the branch instruction.

**Interpretation of `<operand2>`**

`<operand2>` can be interpreted in two different ways, depending on whether the first character is a `#` or an `R`:

- `#` – Use the decimal value specified after the `#`, eg `#25` means use the decimal value 25.
- `Rm` – Use the value stored in register `m`, eg `R6` means use the value stored in register 6.

The available general purpose registers that the programmer can use are numbered 0 to 12.

The AND instruction can be used to help identify if a particular bit in a register contains a 1. The instruction AND R3, R1, #8 will perform the logical bitwise AND operation between the contents of register R1 and the bit pattern 0000 1000 and store the result in register R3. If register R1 has a 1 in bit 3 then the bit pattern for the decimal number 8 will be stored in register R3, otherwise the bit pattern for the decimal number 0 will be stored in register R3.

**Figure 2** and **Figure 3** show examples of this.

**Figure 2** – Example when bit 3 of R1 contains a 1

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| R1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |

Result of

0 1 0 0 1 1 1 0

AND 0 0 0 0 1 0 0 0

is

0 0 0 0 1 0 0 0

So R3 will now contain

0 0 0 0 1 0 0 0

**Figure 3** – Example when bit 3 of R1 contains a 0

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| R1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |

Result of

0 1 0 0 0 1 1 0

AND 0 0 0 0 1 0 0 0

is

0 0 0 0 0 0 0 0

So R3 will now contain

0 0 0 0 0 0 0 0

**0 4 . 2** All even numbers are represented by bit patterns ending with a `0`; all odd numbers are represented by bit patterns ending with a `1`.

Complete the assembly language instruction below to help identify if register `R1` contains an odd number, storing the result of this operation in register `R3`.

**[1 mark]**

```
AND R3, R1,
```

**0 4 . 3** **Figure 4** shows a block of code, written in a high-level language, that is used to find out if the number stored in the variable `A` is even. If the value is even then the number $69_{10}$ is stored in the variable `B`, otherwise the number $79_{10}$ is stored in the variable `B`.

**Figure 4**

```
IF IsEven(A)
  THEN B ← 69
  ELSE B ← 79
ENDIF
```

Write a sequence of assembly language instructions that would perform the same operations as the high-level language code in **Figure 4**.

Assume that register `R1` currently stores the value associated with `A`, that register `R2` stores the value currently associated with `B` and that register `R3` is available for general use, if necessary.

Your answer to question **0 4 . 2** can be reused as part of your answer to this question.

**[6 marks]**

**0 4 . 4** Shade in **one** lozenge to indicate which addressing mode is being used with the second operand in the assembly language instruction `MOV R2, #0`.

**[1 mark]**

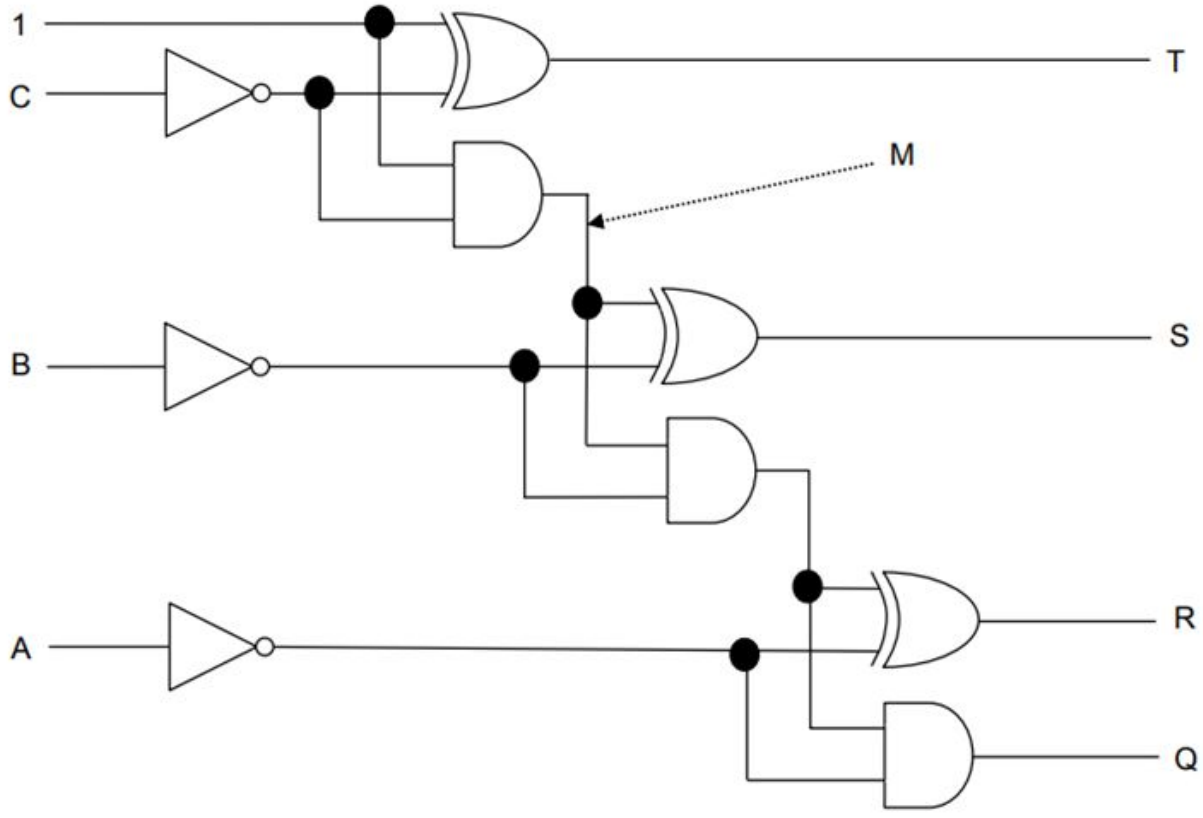
| | | | |
|---|---|---|---|
| **Direct** | ⬭ | **Immediate** | ⬭ |

**0 4 . 5** Explain what a register is.

**[1 mark]**

**Figure 6** shows a logic circuit that might be found inside a processor.

**Figure 6**



**0 5 . 6** The circuit in **Figure 6** can be simplified so that it uses fewer logic gates but still has the same functionality. Changing the design of the circuits used in a processor can improve processor performance. Increasing the number of cores can also improve processor performance.

State **three** other factors that can improve processor performance. For each factor, explain how it will improve processor performance.

**[6 marks]**

**Factor:** _____

**How improves:** _____

_____

_____

**Factor:** _____

**How improves:** _____

_____

_____

**Factor:** _____

**How improves:** _____

_____

June 2017 AS Paper 2

**0 6 . 3** Describe, using full sentences, the steps involved in the Fetch-Execute cycle for the von Neumann architecture. Your description should cover the fetch, decode and execute stages **and** must clearly state which of the three sections each step falls in.

**[6 marks]**

**0 7** **Table 1 – standard AQA assembly language instruction set**. This should be used to answer question parts **0 7 . 1** and **0 7 . 2**

| | |
|---|---|
| `LDR Rd, <memory ref>` | Load the value stored in the memory location specified by `<memory ref>` into register `d`. |
| `STR Rd, <memory ref>` | Store the value that is in register `d` into the memory location specified by `<memory ref>`. |
| `ADD Rd, Rn, <operand2>` | Add the value specified in `<operand2>` to the value in register `n` and store the result in register `d`. |
| `SUB Rd, Rn, <operand2>` | Subtract the value specified by `<operand2>` from the value in register `n` and store the result in register `d`. |
| `MOV Rd, <operand2>` | Copy the value specified by `<operand2>` into register `d`. |
| `CMP Rn, <operand2>` | Compare the value stored in register `n` with the value specified by `<operand2>`. |
| `B <label>` | Always branch to the instruction at position `<label>` in the program. |
| `B<condition> <label>` | Branch to the instruction at position `<label>` if the last comparison met the criterion specified by `<condition>`. Possible values for `<condition>` and their meanings are:<br>`EQ`: equal to `NE`: not equal to<br>`GT`: greater than `LT`: less than |
| `AND Rd, Rn, <operand2>` | Perform a bitwise logical AND operation between the value in register `n` and the value specified by `<operand2>` and store the result in register `d`. |
| `ORR Rd, Rn, <operand2>` | Perform a bitwise logical OR operation between the value in register `n` and the value specified by `<operand2>` and store the result in register `d`. |
| `EOR Rd, Rn, <operand2>` | Perform a bitwise logical XOR (exclusive or) operation between the value in register `n` and the value specified by `<operand2>` and store the result in register `d`. |
| `MVN Rd, <operand2>` | Perform a bitwise logical NOT operation on the value specified by `<operand2>` and store the result in register `d`. |
| `LSL Rd, Rn, <operand2>` | Logically shift left the value stored in register `n` by the number of bits specified by `<operand2>` and store the result in register `d`. |
| `LSR Rd, Rn, <operand2>` | Logically shift right the value stored in register `n` by the number of bits specified by `<operand2>` and store the result in register `d`. |
| `HALT` | Stops the execution of the program. |

**Labels**: A label is placed in the code by writing an identifier followed by a colon (:). To refer to a label the identifier of the label is placed after the branch instruction.

**Interpretation of `<operand2>`**

`<operand2>` can be interpreted in two different ways, depending on whether the first character is a # or an R:

- `#` – use the decimal value specified after the `#`, eg `#25` means use the decimal value 25.
- `Rm` – use the value stored in register `m`, eg `R6` means use the value stored in register 6.

The available general purpose registers that the programmer can use are numbered 0 to 12.

**0 7 . 1** **Figure 5** shows an incomplete assembly language program. The intended purpose of the code is to count from 1 to 10 inclusive, writing the values to memory location 17, which is used to control a motor.

Complete the code in **Figure 5**. You may not need to use all four lines for your solution and you should not write more than one instruction per line.

**[4 marks]**

**Figure 5**

```
            MOV R0, #1
startloop:

            STR R0, 17

            _____________________

            _____________________

            _____________________

            _____________________

endloop:

            HALT
```

**0 7 . 2** `R1` contains the decimal value 7. What value will be contained in `R1` after the instruction below is executed?

```
LSL R1, R1, #2
```

**[1 mark]**

_____

**0 7 . 3** Explain the difference between direct addressing and immediate addressing.

**[1 mark]**

_____

_____

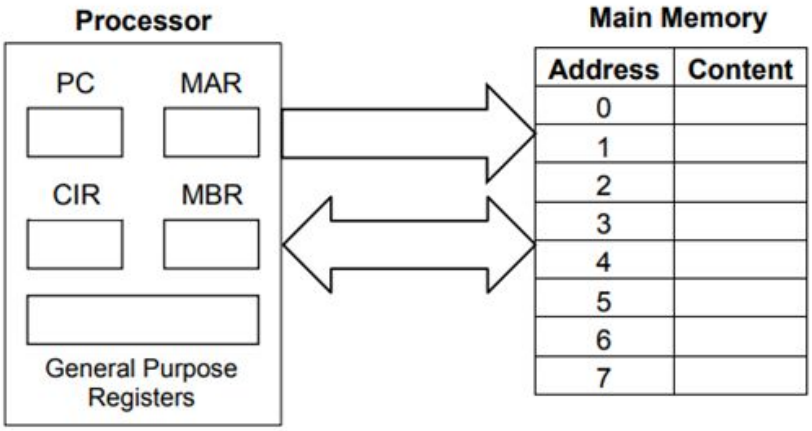
_____

_____

# June 2017 Paper 2

**0 1** **Figure 1** shows some of the internal components of a processor and how the processor is connected to the main memory. The internal connections within the processor are not shown.

**Figure 1**




**0 1 . 1** Describe how an instruction is fetched from main memory during the **fetch stage** of the fetch-execute cycle.

Your description should cover the use of registers and buses, together with the role of main memory.

**[4 marks]**

**0 1 . 2** During the **decode and execute stages** of the fetch-execute cycle the instruction that is being processed is stored in the CIR. Explain why the instruction could **not** be processed directly from the MBR.

**[2 marks]**

**0 5** **Figure 4** shows the structure of an example machine code instruction, taken from the instruction set of a particular processor.

**Figure 4**

| Opcode | | | | | | | Operand(s) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Basic Machine Operation | | | | | | Addressing Mode | | | | | | | | |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |

**0 5 . 1** State the purpose of the operand part of an instruction **and** explain how the addressing mode is related to this.

**[2 marks]**

**Table 1 – standard AQA assembly language instruction set**

| | |
|---|---|
| `LDR Rd, <memory ref>` | Load the value stored in the memory location specified by `<memory ref>` into register `d`. |
| `STR Rd, <memory ref>` | Store the value that is in register `d` into the memory location specified by `<memory ref>`. |
| `ADD Rd, Rn, <operand2>` | Add the value specified in `<operand2>` to the value in register `n` and store the result in register `d`. |
| `SUB Rd, Rn, <operand2>` | Subtract the value specified by `<operand2>` from the value in register `n` and store the result in register `d`. |
| `MOV Rd, <operand2>` | Copy the value specified by `<operand2>` into register `d`. |
| `CMP Rn, <operand2>` | Compare the value stored in register `n` with the value specified by `<operand2>`. |
| `B <label>` | Always branch to the instruction at position `<label>` in the program. |
| `B<condition> <label>` | Branch to the instruction at position `<label>` if the last comparison met the criterion specified by `<condition>`. Possible values for `<condition>` and their meanings are: `EQ`: equal to `NE`: not equal to `GT`: greater than `LT`: less than |
| `AND Rd, Rn, <operand2>` | Perform a bitwise logical AND operation between the value in register `n` and the value specified by `<operand2>` and store the result in register `d`. |
| `ORR Rd, Rn, <operand2>` | Perform a bitwise logical OR operation between the value in register `n` and the value specified by `<operand2>` and store the result in register `d`. |
| `EOR Rd, Rn, <operand2>` | Perform a bitwise logical XOR (exclusive or) operation between the value in register `n` and the value specified by `<operand2>` and store the result in register `d`. |
| `MVN Rd, <operand2>` | Perform a bitwise logical NOT operation on the value specified by `<operand2>` and store the result in register `d`. |
| `LSL Rd, Rn, <operand2>` | Logically shift left the value stored in register `n` by the number of bits specified by `<operand2>` and store the result in register `d`. |
| `LSR Rd, Rn, <operand2>` | Logically shift right the value stored in register `n` by the number of bits specified by `<operand2>` and store the result in register `d`. |
| `HALT` | Stops the execution of the program. |

**Labels**: A label is placed in the code by writing an identifier followed by a colon (:). To refer to a label the identifier of the label is placed after the branch instruction.

**Interpretation of `<operand2>`**

`<operand2>` can be interpreted in two different ways, depending on whether the first character is a # or an R:

- `#` – use the decimal value specified after the `#`, eg `#25` means use the decimal value 25.
- `Rm` – use the value stored in register `m`, eg `R6` means use the value stored in register 6.

The available general purpose registers that the programmer can use are numbered 0 to 12.

**0 5 . 2** Complete the trace table below, **in decimal**, to show how the values stored in the registers and main memory change as the program in **Figure 5** is executed. You may not need to use all of the rows.

**[4 marks]**

| Register Contents | | | | Main Memory Location Contents | | |
|---|---|---|---|---|---|---|
| **R1** | **R2** | **R3** | **R4** | **100** | **101** | **102** |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

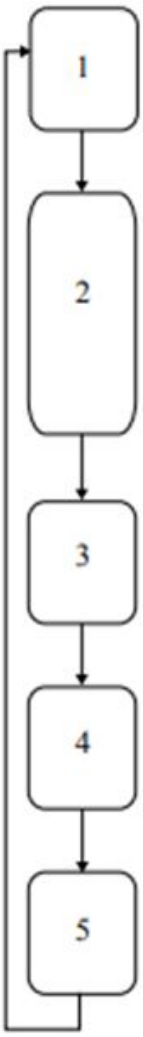
**0 5 . 3** Explain what the assembly language program in **Figure 5** does.

**[1 mark]**

# June 2009 Comp 2

**7** The diagram below shows the fetch-execute cycle. Some of the steps have been described.

**7** (a) Describe the missing steps 1, 2b and 4 using either register transfer notation or a written description. Steps 2a and 2b occur at the same time.




**Step 1:** .....

.....

**Step 2a:** PC ← [PC] + 1
(Increment contents of Program Counter Register)

**Step 2b:** .....

.....

**Step 3:** CIR ← [MBR]
(Transfer contents of Memory Buffer Register into Current Instruction Register)

**Step 4:** .....

.....

**Step 5:** Execute Instruction

*(3 marks)*

**7** (b) What would be the effect on the performance of the computer system of increasing the

**7** (b) (i) width of the data bus? .....

.....

.....

**7** (b) (ii) width of the address bus? .....

.....

.....

**7** (b) (iii) clock speed? .....

.....

.....

*(3 marks)*

# Specimen AS Paper 2

The following registers, listed in alphabetical order, are used in the Fetch-Execute cycle:

- current instruction register (CIR)
- memory address register (MAR)
- memory buffer register (MBR)
- program counter (PC)
- status register (SR).

**0 6 . 2** Describe, **using full sentences**, the steps involved in the Fetch-Execute cycle, and how the registers listed above are used. Your description should cover the fetch, decode and execute phases of the cycle.

**[6 marks]**

**Table 3**

Instructions that can be used in question parts 0 6 . 5 and 0 6 . 6

| | |
|---|---|
| `LDR Rd, <memory ref>` | Load the value stored in the memory location specified by `<memory ref>` into register `d`. |
| `STR Rd, <memory ref>` | Store the value that is in register `d` into the memory location specified by `<memory ref>`. |
| `ADD Rd, Rn, <operand2>` | Add the value specified in `<operand2>` to the value in register `n` and store the result in register `d`. |
| `SUB Rd, Rn, <operand2>` | Subtract the value specified by `<operand2>` from the value in register `n` and store the result in register `d`. |
| `MOV Rd, <operand2>` | Copy the value specified by `<operand2>` into register `d`. |
| `CMP Rn, <operand2>` | Compare the value stored in register `n` with the value specified by `<operand2>`. |
| `B <label>` | Always branch to the instruction at position `<label>` in the program. |
| `B<condition> <label>` | Conditionally branch to the instruction at position `<label>` in the program if the last comparison met the criteria specified by the `<condition>`. Possible values for `<condition>` and their meaning are:<br>• EQ: Equal to.<br>• NE: Not equal to.<br>• GT: Greater than.<br>• LT: Less than. |
| `AND Rd, Rn, <operand2>` | Perform a bitwise logical AND operation between the value in register `n` and the value specified by `<operand2>` and store the result in register `d`. |
| `ORR Rd, Rn, <operand2>` | Perform a bitwise logical OR operation between the value in register `n` and the value specified by `<operand2>` and store the result in register `d`. |
| `EOR Rd, Rn, <operand2>` | Perform a bitwise logical exclusive or (XOR) operation between the value in register `n` and the value specified by `<operand2>` and store the result in register `d`. |
| `MVN Rd, <operand2>` | Perform a bitwise logical NOT operation on the value specified by `<operand2>` and store the result in register `d`. |
| `LSL Rd, Rn, <operand2>` | Logically shift left the value stored in register `n` by the number of bits specified by `<operand2>` and store the result in register `d`. |
| `LSR Rd, Rn, <operand2>` | Logically shift right the value stored in register `n` by the number of bits specified by `<operand2>` and store the result in register `d`. |
| `HALT` | Stops the execution of the program. |

**Interpretation of `<operand2>`**

`<operand2>` can be interpreted in two different ways, depending upon whether the first symbol is a `#` or an `R`:

- `#` - Use the decimal value specified after the `#`, eg `#25` means use the decimal value 25.
- `Rm` - Use the value stored in register `m`, eg `R6` means use the value stored in register 6.

The available general purpose registers that the programmer can use are numbered 0 to 12.

**0 6 . 4** Using the assembly language instruction `CMP R2,R3` explain the term opcode.

**[1 mark]**

0 6 . 5 Explain what immediate addressing is **and** write an example of the use of the `MOV` assembly language instruction, from **Table 3**, that uses immediate addressing.

**[2 marks]**

0 6 . 6 **Figure 2** shows a block of program code, written in a high-level language.

**Figure 2**

```
IF X = 5
  THEN B ← 10
END IF
```

Write a sequence of assembly-language instructions that would perform the same operations as the program code in **Figure 2**. Assume that register `R1` currently stores the value associated with `X`, register `R2` stores the value currently associated with `B` and that register `R3` is available for general use, if necessary.

**[4 marks]**

**Table 2**

| | |
|---|---|
| `LDR Rd, <memory ref>` | Load the value stored in the memory location specified by `<memory ref>` into register `d`. |
| `STR Rd, <memory ref>` | Store the value that is in register `d` into the memory location specified by `<memory ref>`. |
| `ADD Rd, Rn, <operand2>` | Add the value specified in `<operand2>` to the value in register `n` and store the result in register `d`. |
| `SUB Rd, Rn, <operand2>` | Subtract the value specified by `<operand2>` from the value in register `n` and store the result in register `d`. |
| `MOV Rd, <operand2>` | Copy the value specified by `<operand2>` into register `d`. |
| `CMP Rn, <operand2>` | Compare the value stored in register `n` with the value specified by `<operand2>`. |
| `B <label>` | Always branch to the instruction at position `<label>` in the program. |
| `B<condition> <label>` | Conditionally branch to the instruction at position `<label>` in the program if the last comparison met the criteria specified by the `<condition>`. Possible values for `<condition>` and their meaning are:<br>• EQ: Equal to.<br>• NE: Not equal to.<br>• GT: Greater than.<br>• LT: Less than. |
| `AND Rd, Rn, <operand2>` | Perform a bitwise logical AND operation between the value in register `n` and the value specified by `<operand2>` and store the result in register `d`. |
| `ORR Rd, Rn, <operand2>` | Perform a bitwise logical OR operation between the value in register `n` and the value specified by `<operand2>` and store the result in register `d`. |
| `EOR Rd, Rn, <operand2>` | Perform a bitwise logical exclusive or (XOR) operation between the value in register `n` and the value specified by `<operand2>` and store the result in register `d`. |
| `MVN Rd, <operand2>` | Perform a bitwise logical NOT operation on the value specified by `<operand2>` and store the result in register `d`. |
| `LSL Rd, Rn, <operand2>` | Logically shift left the value stored in register `n` by the number of bits specified by `<operand2>` and store the result in register `d`. |
| `LSR Rd, Rn, <operand2>` | Logically shift right the value stored in register `n` by the number of bits specified by `<operand2>` and store the result in register `d`. |
| `HALT` | Stops the execution of the program. |

**Interpretation of `<operand2>`**

`<operand2>` can be interpreted in two different ways, depending upon whether the first symbol is a `#` or an `R`:

- `#` - use the decimal value specified after the `#`, eg `#25` means use the decimal value 25.
- `Rm` - use the value stored in register `m`, eg `R6` means use the value stored in register 6.

The available general purpose registers that the programmer can use are numbered 0 to 12.

**Question 6 continues on the next page**

**0 6** **Figure 3** shows the structure of an example machine code instruction, taken from the instruction set of a particular processor.

**Figure 3**

| Opcode | | | | | | | Operand(s) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Basic Machine Operation | | | | | | Addressing Mode | | | | | | | | |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |

**0 6 . 1** How many different basic machine operations could be supported by the instruction set of the processor used in the example in **Figure 3**?

**[1 mark]**

**Figure 4** shows an assembly language program together with the contents of a section of the main memory of the computer that the program will be executed on.

The assembly language instruction set that has been used to write the program is listed in **Table 2**.The lines of the assembly language program have been numbered to help you answer question parts 0 6 . 2 to 0 6 . 4.

**Figure 4**

| Line | Command |
|---|---|
| 1 | `MOV R2, #100` |
| 2 | `LDR R3, 101` |
| 3 | `ADD R2, R2, R3` |
| 4 | `LSL R3, R2, #1` |
| 5 | `HALT` |

| Memory Address (in decimal) | Main Memory Contents (in decimal) |
|---|---|
| 100 | 23 |
| 101 | 10 |
| 102 | 62 |
| 103 | 18 |

**0 6 . 2** What value will be stored in register `R2` immediately after the command in line 1 has been executed?

**[1 mark]**

**0 6 . 3** What value will be stored in register `R2` immediately after the program has executed the commands from line 1 through to line 3?

**[1 mark]**

**0 6 . 4** What value will be stored in register `R3` after the complete program has finished executing?

**[1 mark]**