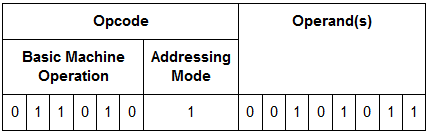
**Q1.**

**Figure 1** shows the structure of an example machine code instruction, taken from the instruction set of a particular processor.

**Figure 1**

****

(a)     How many different basic machine operations could be supported by the instruction set of the processor used in the example in **Figure 1**?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

**Figure 2** 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 1**.The lines of the assembly language program have been numbered to help you answer question parts (**b**) to (**d**)

**Figure 2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Line** | **Command** |  | **Memory Address (in decimal)** | **Main Memory Contents (in decimal)** |
| **1** | LDR  R2,   #100 |  |
| **2** | LDR  R3,   101 |  |
| **3** | ADD  R2,   R2,   R3 |  | 100 | 23 |
| **4** | LSL  R3,   R2,   #1 |  | 101 | 10 |
| **5** | HALT |  | 102 | 62 |
|  |  |  | 103 | 18 |

(b)     What value will be stored in register R2 immediately after the command in line 1 has been executed?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

(c)     What value will be stored in register R2 immediately after the program has executed the commands from line 1 through to line 3?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

(d)     What value will be stored in register R3 after the complete program has finished executing?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Table 1**

|  |  |
| --- | --- |
| 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:      •    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 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.

Programs written in a high-level language can be compiled or interpreted.

Companies that develop computer programs to sell usually compile the final version of a program before distributing it to customers.

(e)     Explain why the final version of a computer program is usually translated using a compiler.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(2)**

(f)     The JavaScript programming language can be used to write programs that are executed in a web browser on any Internet user’s computer.

Explain why programs written in the JavaScript language, to be executed in a web browser, are interpreted rather than compiled.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(2)**

**(Total 8 marks)**

**Q2.**

(a)     Assembly language is considered to be a low-level language. Which other type of language is also considered to be a low-level language?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

The table shows the standard AQA assembly language instruction set that should be used to answer question parts **(b)** and **(c)**.

|  |  |
| --- | --- |
| 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 #, e.g. #25 means use the decimal value 25.

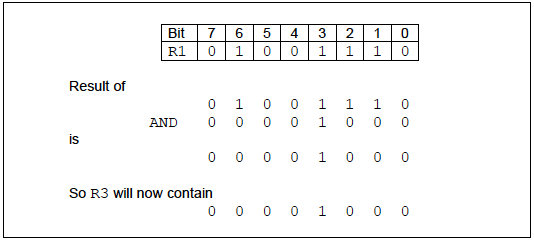
•        Rm – Use the value stored in register m, e.g. R6 means use the value stored in register 6.

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

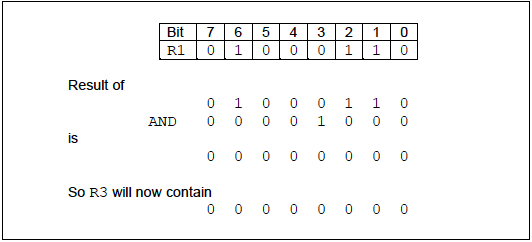
The AND instruction can be used to help identify if a particular bit in a register contains a 1. The instruction AND R3, R1, #8 will perform the logical bitwise AND operation between the contents of register R1 and the bit pattern 0000 1000 and store the result in register R3. If register R1 has a 1 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 1** and **Figure 2** show examples of this.

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



**Figure 2** – Example when bit 3 of R1 contains a 0



(b)     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.

AND R3, R1, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

(c)     **Figure 3** 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 6910 is stored in the variable B, otherwise the number 7910 is stored in the variable B.

**Figure 3**

|  |  |
| --- | --- |
|  | 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 3**.

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 **(b)** can be reused as part of your answer to this question.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(6)**

(d)     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)**

(e)     Explain what a register is.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

**(Total 10 marks)**