# Homework 4 Assembly language Answers

# The assembly code instruction formats in Table 1 are used in the questions in this homework.

1. Examine the following assembly code.

 LDR R1, 101

 LDR R2, 102

 label1:

 SUB R1, R1, R2

 CMP R1, #0

 BGT label1

 BEQ label2

 ADD R1, R1, R2

 label2:

 STR R1, 103

 (a) If memory location 101 contains the number 23 and memory location 102 contains the number 7, what will be the contents of memory location 103 after the execution of the code? [2]

 Answer: 2 (See trace table below)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **R1** | **R2** | **101** | **102** | **103** |
|  |  | 23 | 7 |  |
| 23 |  |  |  |  |
|  | 7 |  |  |  |
| 16 |  |  |  |  |
| 9 |  |  |  |  |
| 2 |  |  |  |  |
| -5 |  |  |  |  |
| 2 |  |  |  | 2 |

 (b) Explain what the instruction SUB R1, R1, R2 does the first time around the loop starting at label1. [1]

 It subtracts 7 from the contents of R1, which is 23 and stores the result (16) back in R1.

 (c) How many times is the loop performed? [1]

 4 times

 (d) What does the program do? [1]

 It finds the remainder when the number in 101 is divided by the number in 102

2. (a) Write assembly code instructions to compare the contents of register 1 and register 2 and branch to label1 if they are equal. [2]

 CMP R1, R2 ;compare contents of R1 and R2

 BEQ label1 ;branch if equal to label1

 *or,*

 EOR R3, R1, R2 ;perform exclusive OR between R1 and R2, result in R3

 CMP R3, #0 ;R3 will be 0 if R1 and R2 are equal

 BEQ label1 ;branch if equal to label1

(b) Explain with the aid of examples the difference between **immediate** and **direct** addressing.

 [2]

 With immediate addressing, the operand holds the actual value to be used in the operation.

 Example:

 ADD R1, R2, #78 ; add the number 78 to contents of R2 and store in R1

 With direct addressing, the operand holds the address of the value to be used in the operation.

 LRD R1, 78 ; load the value stored in memory address 78

 ; and store in R1

 (or any other suitable examples)

3. Write the assembly code instructions which are equivalent to the following high-level code. Comment each line of code.

 x 🡨 0

 WHILE x < 1000

 x = x + 1

 ENDWHILE [6]

 MOV R1, #0 ; initialise R1 to 0 to hold x

 label1:

 ADD R1, R1, #1 ; add 1 to R1 and store result back in R1

 CMP R1, #1000 ; compare R1 with 1000

 BLT label1 ; branch if R1 < 1000

 (continue) ; R1 is equal to 1000

 (4 for correct instructions, 2 for explicit comments)

4. In a particular computer, characters are represented by 8-bit patterns. The codes for uppercase letters are from 0100 0001 for A to 0101 1010 for Z. The codes for lowercase letters are from 0110 0001 for a to 0111 1010 for z. Give an appropriate mask and logical operation which will:

 (a) change any uppercase letter to its lowercase equivalent [2]

 Mask: 0010 0000 operation XOR

 (The 3rd bit from the left is the only bit which changes. XOR –ing with 0 leaves the original bit as it is, XOR-ing with 1 flips the bit.)

 (b) change any lowercase letter to its uppercase equivalent [2]

 Mask: 0010 0000 operation XOR (same as above)

5. An 8-bit word contains the bit pattern 10110010.

 State the contents of the word after a logical right shift of 3 bits. [1]

 00010110 [Total 20 Marks] [Total 20 Marks]

**Table 1: 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, <operand>  | Add the value specified in <operand> to the value in register n and store the result in register d.  |
| SUB Rd, Rn, <operand>  | Subtract the value specified by <operand> from the value in register n and store the result in register d.  |
| MOV Rd, <operand>  | Copy the value specified by <operand> into register d.  |
| CMP Rn, <operand>  | Compare the value stored in register n with the value specified by <operand>.  |
| 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, <operand>  | Perform a bitwise logical AND operation between the value in register n and the value specified by <operand> and store the result in register d.  |
| ORR Rd, Rn, <operand>  | Perform a bitwise logical OR operation between the value in register n and the value specified by <operand> and store the result in register d.  |
| EOR Rd, Rn, <operand>  | Perform a bitwise logical exclusive or (XOR) operation between the value in register n and the value specified by <operand> and store the result in register d.  |
| MVN Rd, <operand>  | Perform a bitwise logical NOT operation on the value specified by <operand> and store the result in register d.  |
| LSL Rd, Rn, <operand>  | Logically shift left the value stored in register n by the number of bits specified by <operand> and store the result in register d.  |
| LSR Rd, Rn, <operand>  | Logically shift right the value stored in register n by the number of bits specified by <operand> and store the result in register d.  |
| HALT  | Stops the execution of the program.  |

<operand> 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 #, 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 7.