# Worksheet 4 Assembly language

**Use the instructions in Table 1 on the next page for all questions on this worksheet**

**Task 1**

1. A positive number is held in memory location 201. Write assembly code instructions which put the negative of the number in location 202. [4]

2. Write assembly code instructions to take three numbers held in locations 201, 202 and 203 and store them back in the same locations in reverse order. [4]

3. Three numbers are held in locations 201, 202 and 203. Write assembly code instructions to store the maximum of the three numbers in location 300. [5]

4. Assume that numbers can be input by a user into Register 0 using the instruction

INP R0

(a) Ten numbers are input by the user. Write assembly code instructions to add the ten numbers as they are input and store the result in location 300. [5]

(b) Ten integers are input by the user. Write assembly code instructions to count the number of integers that are greater than or equal to 30. The result should be stored in location 300. [6]

**Task 2**

1. Write assembly code instructions to find whether a number held in location 200 is odd or even. [5]

2. A bit pattern held in R0 represents 8 switches numbered 1 to 8 (left to right).

(a) Write assembly code instructions to initialise the switches to zero, and then turn on switches 1,3,5 and 7. [2]

(b) Assume you do not know the state of the switches in R0. Write an assembly code instruction to turn off any switches that are on, and vice versa. [1]

3. What will be the effect of performing an XOR operation on an operand with itself [1]

Give an example. [1]

**Task 3**

1. Assume R0 contains an 8-bit positive integer. Using logical shifts, compare and branch operations, write assembly code statements to branch to LABEL1 if the integer represents an even number, otherwise continue to the next statement. [5]

Test your program by tracing the contents of R0 and any other registers used. [3]

Can you think of another way of testing whether the number is even? [2]

**Table 1: Assembly language instructions**

|  |  |
| --- | --- |
| 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.