**TASK 1 – Low-level programming**

Low-level programming is a type of programming language that uses op codes and operands to create instructions.

The table shows part of the instruction set for a processor that has one general purpose register, the Accumulator (ACC), and an Index Register (IX).

| Instruction | | Explanation |
|---|---|---|
| **Op code** | **Operand** | |
| LDM | #n | Immediate addressing. Load the number n to ACC. |
| LDD | <address> | Direct addressing. Load the contents of the location at the given address to ACC. |
| LDI | <address> | Indirect addressing. The address to be used is at the given address. Load the contents of this second address to ACC. |
| LDX | <address> | Indexed addressing. Form the address from <address> + the contents of the Index Register. Copy the contents of this calculated address to ACC. |
| LDR | #n | Immediate addressing. Load the number n to IX. |
| STO | <address> | Store the contents of ACC at the given address. |
| STX | <address> | Indexed addressing. Form the address from <address> + the contents of the Index Register. Copy the contents of ACC to this calculated address. |
| ADD | <address> | Add the contents of the given address to ACC. |
| INC | <register> | Add 1 to the contents of the register (ACC or IX). |
| DEC | <register> | Subtract 1 from the contents of the register (ACC or IX). |
| JMP | <address> | Jump to the given address. |
| CMP | <address> | Compare the contents of ACC with the contents of <address>. |
| CMP | #n | Compare the contents of ACC with number n. |
| JPE | <address> | Following a compare instruction, jump to <address> if the compare was True. |
| JPN | <address> | Following a compare instruction, jump to <address> if the compare was False. |
| AND | #n | Bitwise AND operation of the contents of ACC with the operand. |
| AND | <address> | Bitwise AND operation of the contents of ACC with the contents of <address>. |
| XOR | #n | Bitwise XOR operation of the contents of ACC with the operand. |
| XOR | <address> | Bitwise XOR operation of the contents of ACC with the contents of <address>. |
| OR | #n | Bitwise OR operation of the contents of ACC with the operand. |
| OR | <address> | Bitwise OR operation of the contents of ACC with the contents of <address>. <address> can be an absolute address or a symbolic address. |
| LSL | #n | Bits in ACC are shifted n places to the left. Zeros are introduced on the right hand end. |
| LSR | #n | Bits in ACC are shifted n places to the right. Zeros are introduced on the left hand end. |
| IN | | Key in a character and store its ASCII value in ACC. |
| OUT | | Output to the screen the character whose ASCII value is stored in ACC. |
| END | | Return control to the operating system. |

**TASK 1.1**

Write assembly language program code that allows a user to input 5 characters. The characters are not stored.

| Label | Op code | Operand | Comment |
|---|---|---|---|
| | LDM | #0 | // initialise COUNT to 0 |
| | STO | COUNT | |
| LOOP: | IN | | // input character |
| | LDD | COUNT | // increment COUNT |
| | INC | ACC | |
| | STO | COUNT | |
| | CMP | MAX | // is COUNT = MAX ? |
| | JPN | LOOP | // jump to LOOP if FALSE |
| | END | | // end program |
| MAX: | 5 | | |
| COUNT: | | | |

**TASK 1.2**

Discuss the purpose of the Index Register and how it can be used to access consecutive memory locations.

An index register in a computer's CPU is a processor register used for modifying operand addresses during the run of a program, typically for doing array operations on consecutive memory loactions.

The contents of an index register (displacement) is added to an immediate address (one that is part of the instruction itself or base address) to form the "effective" address of the actual data (operand) .

This IX is used to access the consecutive memory locations as in an array.

**TASK 1.3**

Write assembly language program code that adds the values stored in four consecutive memory locations starting at `NUMBER` using the Index Register.

Store the final total value in memory location `TOTAL`.

| Label | Op code | Operand | Comment |
|---|---|---|---|
| | **LDR** | **#0** | `// initialise Index Register to 0` |
| `LOOP:` | **LDX** | **NUMBER** | `// load value from NUMBER + contents of Index Register` |
| | **ADD** | **TOTAL** | `// add value to TOTAL` |
| | **STO** | **TOTAL** | |
| | **INC** | **IX** | `// increment Index Register` |
| | **LDD** | **COUNT** | `// increment COUNT` |
| | **INC** | **ACC** | |
| | **STO** | **COUNT** | |
| | **CMP** | **MAX** | `// is COUNT = MAX ?` |
| | **JPN** | **LOOP** | `// jump to LOOP if FALSE` |
| | **END** | | `// end program` |
| `MAX:` | `4` | | |
| `COUNT:` | `0` | | |
| `NUMBER:` | `23` | | |
| | `17` | | |
| | `38` | | |
| | `13` | | |
| `TOTAL:` | `0` | | |

**TASK 1.4**

The assembly language instruction set given has the op code `STX`. Discuss the purpose of this op code.

STX, Store Indexed: It means the content of the ACC, accumulator register, will be saved at the actual address that is formed by adding base address (given in instruction operand) and content of the IX, index register,(displacement).

For Example:
IX = 3
MYADD = 300
ACC = 200
THEN:
STX MYADD, will save the ACC content 200 at address 300 + 3 = 303.

**TASK 1.5**

Amend your solution to **TASK 1.1** to allow the program to store each of the characters input into separate, consecutive memory locations starting at the memory locations labelled `CHARACTER`.

| Label | Op code | Operand | Comment |
|---|---|---|---|
| | LDM | #0 | // initialise COUNT to 0 |
| | STO | COUNT | |
| | LDR | #0 | // initialise Index Register to 0 |
| LOOP: | IN | | // input character |
| | STX | CHARACTER | // store ACC to CHARACTER + contents of Index Register |
| | INC | IX | // increment Index Register |
| | LDD | COUNT | |
| | INC | ACC | // increment COUNT |
| | STO | COUNT | |
| | CMP | MAX | // is COUNT = MAX ? |
| | JPN | LOOP | // jump to LOOP if FALSE |
| | END | | // end program |
| MAX: | 5 | | |
| COUNT: | | | |
| CHARACTER: | | | |
| | | | |
| | | | |
| | | | |
| | | | |