Prac 04: SIMP Sorter

Task

Write a SIMP program to sort numbers in ascending order. The SIMP machine is described in Appendix C of the Brookshear text.

Preparation

Before coming to your prac class, install the SIMP machine simulator on your laptop or U drive. To install the app, download **`SIMP.jar`** from FLO and move it to a convenient place. To run the simulator, double click the file.

To write a program for the simulator, use a spreadsheet, text editor or similar application to prepare a list of SIMP instruction codes in the required format, then copy the text and paste it into the data input window of the simulator. You'll also need to provide input lines to initialise **`PC`** to the program's starting address and to load suitable test values into memory.

To run the program, clear and load the memory using the buttons at the bottom of the simulator window, then either run or step it to test that it operates correctly.

Checkpoint 1: Swap

1. Write a SIMP program that will swap the values stored in memory locations **`F0`** and **`F1`**. The idea is to load two registers with the values from memory, then store the register values back to memory "the other way round". For example, if you load register 1 from **`F0`** and register 2 from **`F1`**, you would store **`R1`** back to **`F1`** and **`R1`** to **`F0`**.
2. Test your code with the SIMP simulator, using suitable values in **`F0`** and **`F1`**. For example if **`F0`** contains **`07`** and **`F1`** contains **`03`**, the final value in **`F0`** should be **`03`**, and the final value in **`F1`** should be **`07`**.
3. To earn the checkpoint, complete Question 1 of the Prac 4 quiz on FLO; there is no need to show your work to a demonstrator, although of course they are available to help. You will need to copy and paste text that defines your program code and the initial value of the **`PC`**; a variety of test values will be added automatically.

Checkpoint 2: Sort 2

1. Extend your program so that it examines the values in **`F0`** and **`F1`** and swaps them if **`F0`** is larger than **`F1`**; otherwise it should leave them unchanged. The final result should thus be that the values in **`F0`** and **`F1`** are in ascending numeric order.
2. To compare the values, subtract **`F1`** from **`F0`** and test to see if the result is positive or negative. If it is positive (indicating that **`F0`** was the larger number), you will need to swap the values. Otherwise, jump over the swap code.
3. Test your program with the simulator, using suitable values for **`F0`** and **`F1`**. Note that because the program makes use of subtraction by negation, it will only produce correct results for positive values (values in the range **`00`** to **`7F`**).
4. To earn the checkpoint, complete Question 2 of the Prac 4 quiz on FLO.

Checkpoint 3: Sort 3

1. Extend the program so that it sorts the 3 values in **F0**, **F1**, and **F2** into ascending order. A 3-value sort can be performed by performing three 2-value sorts. Start by sorting the first and second values, then sort the second and third values, then finally sort the first and second values again. For example, consider this scenario:

```
4 5 3      // initial values
4 5 3      // sort first pair (no swap needed)
4 3 5      // sort second pair (swapped)
3 4 5      // sort first pair again (swapped)
```

2. Test your code using the simulator, with suitable values in **F1**, **F2**, and **F3**. Make sure your code will work for numbers in any initial order.
3. To earn the checkpoint, complete Question 3 of the Prac 4 quiz on FLO.

To Go Further (no extra marks): Sort n

1. Extend the program so that it will sort the list of values in **F0** and subsequent memory cells, up to and including the first cell that contains **00**.
2. The simplest approach (a variant of the "bubble sort" algorithm) is to write code that sorts successive pairs of numbers from the beginning of the list until it finds (and swaps) a pair with a second value of **00**, then keep repeating the whole process until the value of the first element in the list is **00**. For example

```
4 5 3 2 6 0  // first sweep through
4 3 5 2 6 0
4 3 2 5 6 0
4 3 2 5 6 0
4 3 2 5 0 6  // 0 indicates end of sweep
3 4 2 5 0 6  // repeat; last item is now good
3 2 4 5 0 6
3 2 4 5 0 6
3 2 4 0 5 6  // end of sweep
2 3 4 0 5 6  // repeat
2 3 4 0 5 6
2 3 0 4 5 6
2 3 0 4 5 6  // repeat
2 0 3 4 5 6
0 2 3 4 5 6  // repeat
0 2 3 4 5 6  // first item is 0; stop
```

   To implement the algorithm, store the memory address of the first number in the pair (a "pointer" value) in a register, then use the **GET** and **PUT** instructions (not in the original Brookshear machine) to access memory at that address. To proceed to the next pair, increment the pointer register. To repeat the process, set the pointer back to the beginning of the list.

```
GET (opcode D, arguments R0T):
    fetch into reg R the memory cell whose address is in reg T
PUT (opcode E, arguments R0T):
    store reg R into the memory cell whose address is in reg T
```

3. Test your program by completing Question 4 of the Prac 4 quiz on FLO.