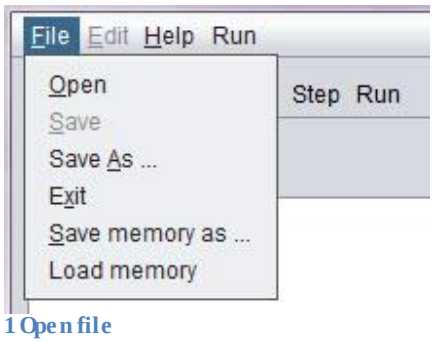
GUI:

The graphic interface must include the following elements:
- Window (or frame) editing to introduce programs in assembly language. Buttons to save disk drive or loaded from an assembly program.

1. The file will be opened as in the following image, using the File/Open menu item/Sub menu item.



1 Open file

- Window (or frame) to display processor registers (PC, SP and RA) and outcome indicators (C, N, Z and V). These records will be updated either by the execution of the instructions or because the user modifies the content. You should view/ edit both binary and hexadecimal.

- Two windows (or frames) to display the computer's memory (256 bytes). This allows the student displayed at the same time the bottom of memory where your program into machine code, and in the other window another address range, which may have data or be the stack. Both memory addresses and their contents are displayed in hexadecimal. It must be possible to modify the contents of a memory address. The way it is now does not quite like, but not if it is possible to make a memory location by clicking on its value and changing it is changed.

- A seven-segment display two digits to display the output of 0x01 OUT instruction (is as it is in the current simulator)

- A peripheral output to your liking, Something chart, which will send a byte or two and change color, shape or position. For example, a needle that moves right or left depending on the value that you send it ...

- A battery of 8 switches, (as an example you can see, but need not be so https://www.pantechsolutions.net/components/switches/dip-switch) to generate the input to the instruction IN 0x00. The user must be able to change the position of each of the switches (on / off) with the mouse

- A hexadecimal as having the current simulator keyboard, but instead of an OK button, you will have something like a push button (https://electrosome.com/wp-content/uploads/2012/12/Push-Button-Switch.jpg).

- A panel with buttons (or bar, although I prefer panel with large buttons so you can read the function of each button) with the following buttons:
Assemble, Step (Step by Step Run) Run (Uninterruptible Execution), Reset PC (Set to 0 the PC), Reset RA (Reset the RA), Reset SP (Reset the SP), Reset memory (Reset all memory) and RESET (Resets all of the above). STOP button

Behavior of the different elements:

-buttons Reset PC, RA, SP and memory.
These buttons should reset the corresponding record (or all memory). The outcome indicators are reset in conjunction with RA or when reset everything.

Joining -button
Once introduced into an assembly program editor, the simulator must analyze it to translate into machine code. If an error is detected, for example, the instruction does not exist or is missing an operand, display an error message and indicate where the error occurred. It takes no great sophistication, as the language is simple and regular.
If no errors are loaded into memory the result of code the program into machine code.

- Step Button
Execute the instruction pointed to by the PC and stop updating all the graphics and state of the computer elements.

- Run Button
Execute the instruction pointed to by the PC, update the machine status and graphic elements, and continue with the next instruction. The execution will stop when it reaches the Stop button or press the STOP instruction.

- Stop Button
Stops program execution.

- Records and memory must change according to the execution of instructions (and if the user forces a value, of course)

- It would be very interesting that when a memory location is changed in the second window that shows the memory will be displayed, if it is not already, the memory area around the modified position.

- When a step execution is done, it should light or at least an arrow pointing instruction to be executed. If you can do both in the assembly code and source code would be fine.

Instructions to run:

In addition to the instructions in the documentation, including byte (which is not an instruction, is a thing of assembler) are two more who would want will implement instructions. And CLEAR are COMPAREP PORT PORT, but these'll talk later, when you start scheduling the execution of instructions.

The code to update indicators C, N, Z and V you'll pass, because although it is not complex is not trivial.