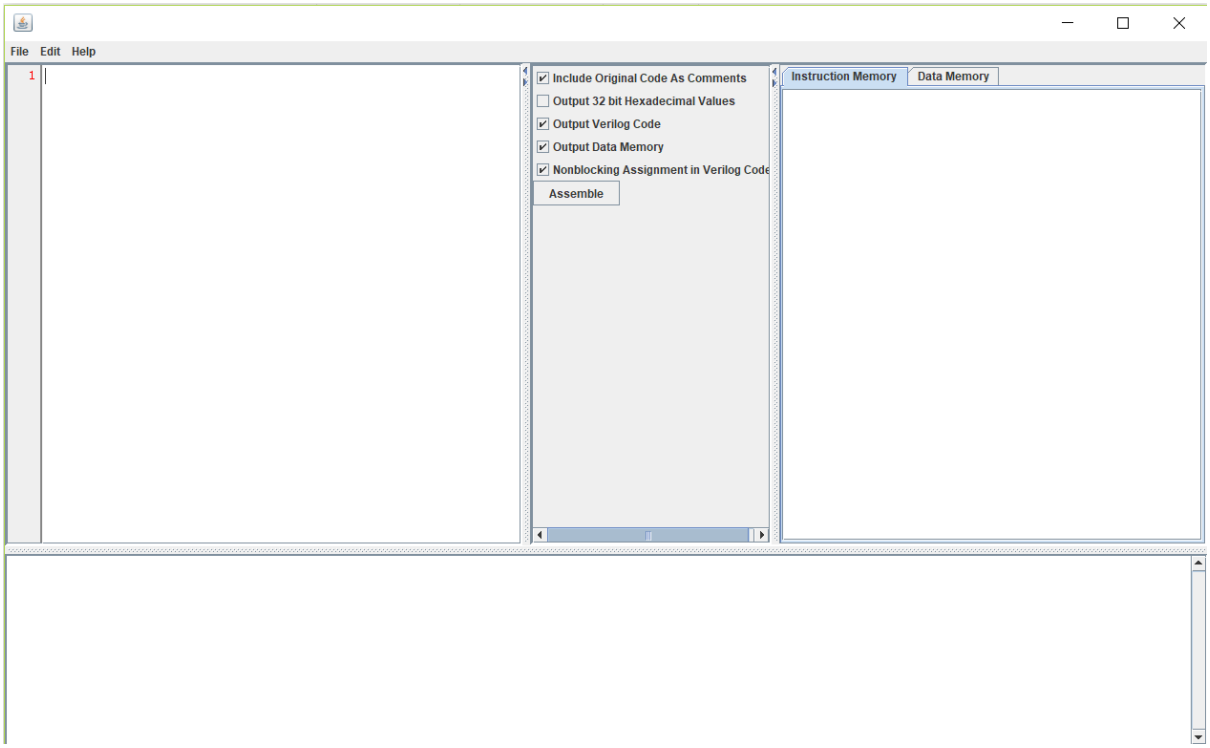
Tutorial – MipsHelper369

MipsHelper369 is an assembler tool for generating Verilog-based instruction and data memories rapidly for a given MIPS assembly code. It was originally implemented by Nathaniel Sema while he was taking the ECE-369A. Later, Muneeb Mateen Ahmed fixed the known bugs of the existing MipsHelper369 and created GUI support. The latest version is contained in the folder **MipsHelper2018**.

This tutorial shows how to use the MipsHelper369 to convert an assembly code to Instruction (machine code) and Data memory contents.

Step 1: Familiarize with the interface:

To launch the MipsHelper369, unzip the delivered "MipsHelper2018.zip". In the extracted MipsHelper2018 folder, launch 'MipsHelper2018/binary/src/MipsHelper369.jar'. This should invoke the MipsHelper369 user interface as the following figure.



The tools on the top left corner are:
File => This button can be used to open and load assembly codes and save the output files.
Edit => Editing the already loaded assembly code in MipsHelper369. Currently the only editing option is 'Clear'.
Help => This option contains important information regarding MIPS Instruction Sets that might come handy to students in understanding the machine code translation from assembly code, and better plan the design of datapath.

The panel in the middle contains few options that decide the format of data in generated output files.

- **Include Original Code As Comments:** If selected, the 32-bit instructions will have the corresponding assembly code added as a comment in the same line. This might come handy in debugging.
- **Output 32-bit Hexadecimal Values:** If selected, the 32-bit instructions will be written in a hex format (e.g. 0x89ABCDEF). If not selected, the instructions will be written in binary format (e.g. 10001001101010111100110111101111).
- **Output Verilog Code:** If selected, each instruction will be assigned to the sequential index in instruction memory. For example, the translated instructions should look as-

  memory[0] = 32'b10001001101010111100110111101111
  memory[1] = 32'b10011011101010111100110111101111
  …… and so on.

  Therefore, the instruction memory in Verilog code can be easily initialized by copying and pasting the generated file content.
- **Output Data Memory:** If selected, the tool will also output the contents of Data memory as a file. This file can be used to initialize the data memory of datapath.
- **Nonblocking Assignment in Verilog Code:** If selected, the output Verilog code for initializing memory contents will contain nonblocking assignments (<=).

  For example, the translated instructions will look as-

  memory[0] <= 32'b10001001101010111100110111101111
  memory[1] <= 32'b10011011101010111100110111101111
  …… and so on.
- **Assemble:** After selecting suitable options from above mention list, hit this button to generate Instruction and Data memory conents.

The right most panel has two tabs- **Instruction Memory** and **Data Memory**. These tabs show the contents of Instruction and Data memory after conversion.

The bottom panel is for delivering messages (error, statistics, etc.) related to conversion.

Step 2: Sample conversion using 'sum_of_array.s' file:

In this step, the 'sum_of_array.s' file will be loaded into MipsHelper369 and output files will be generated.

The 'sum_of_array.s' file contains the following Data Memory contents:

```
N:    .word 3                          # 'N' is the address which contains the loop count, 5
X:    .word -2, -4, 7                  # 'X' is the address of the 1st element in the array to be added
SUM:   .word 0                         # 'SUM' is the address that stores the final sum
str:  .asciiz "The sum of the array is = "
```

These memory contents will be generated as **Data Memory**.

The remaining portion of the 'sum_of_array.s' that is assembly code, will be converted into corresponding machine code for initializing the Instruction Memory, and will be displayed under the **Instruction Memory** window.

The conversion steps are as following-

(a) Click **File** → **Open** and select the .s file to be converted. In this case, the 'sum_of_array.s' file is loaded.

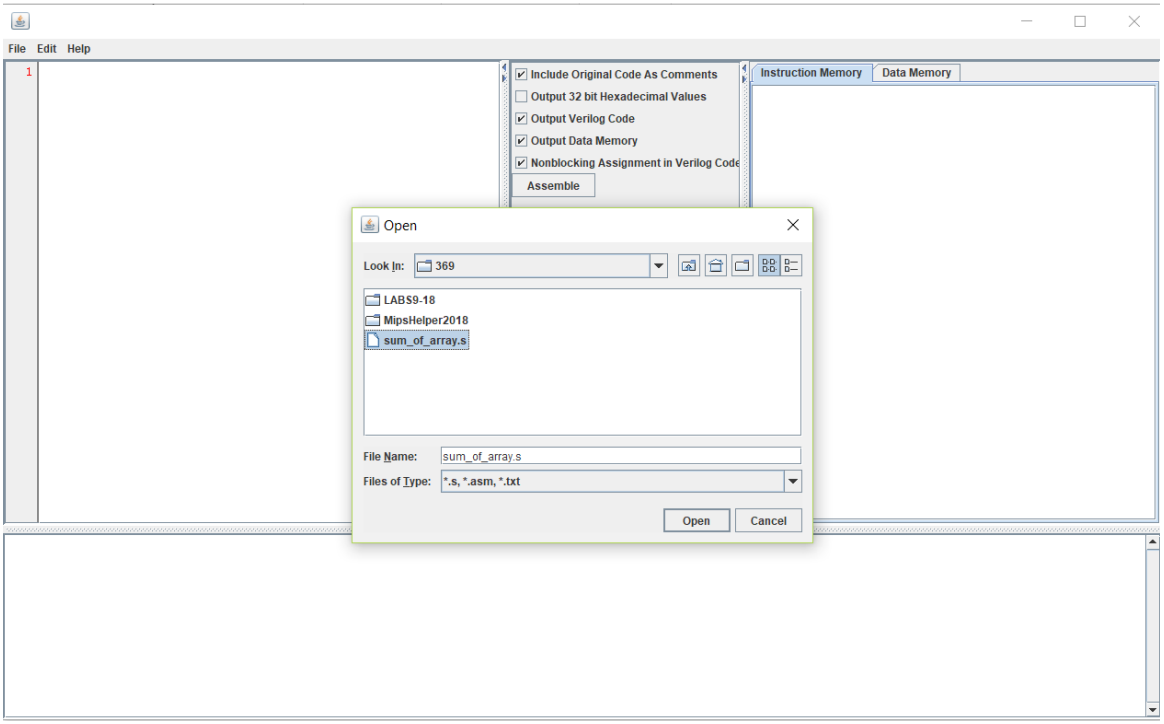


Fig: Opening and loading a .s file.

(b) Select the proper output format options from the middle panel, and hit **Assemble**.

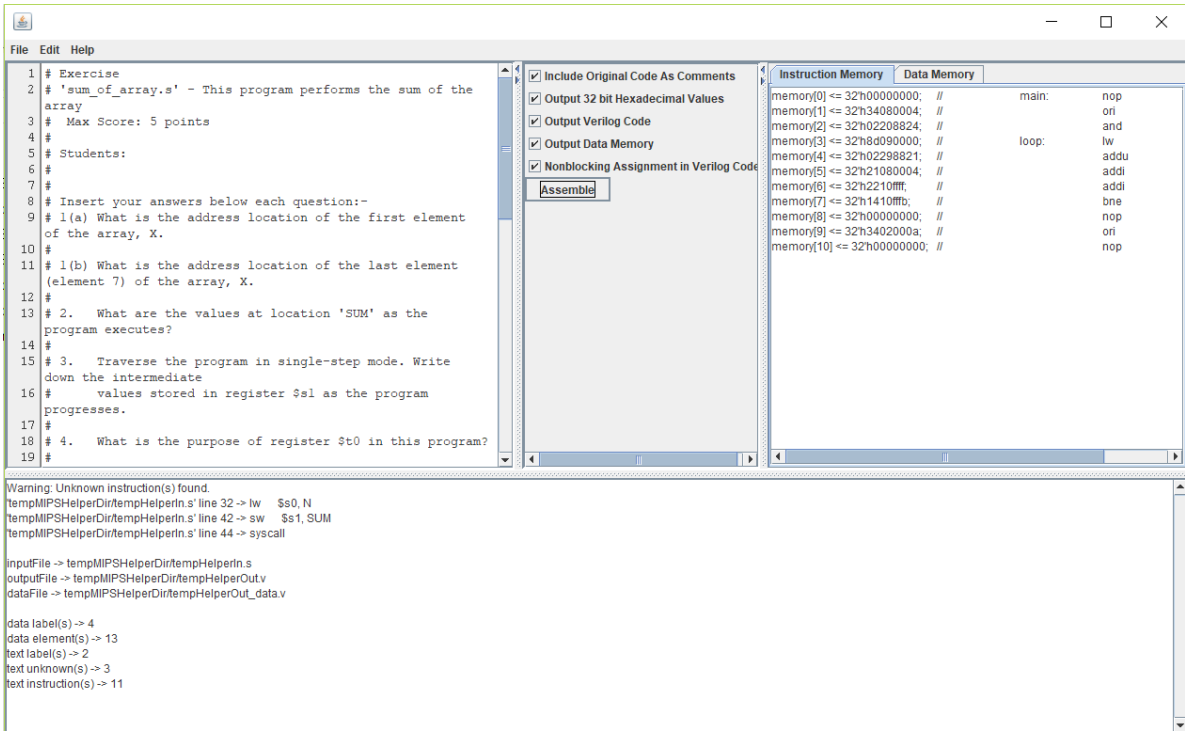


Fig: Generated Instruction Memory contents after assembling.

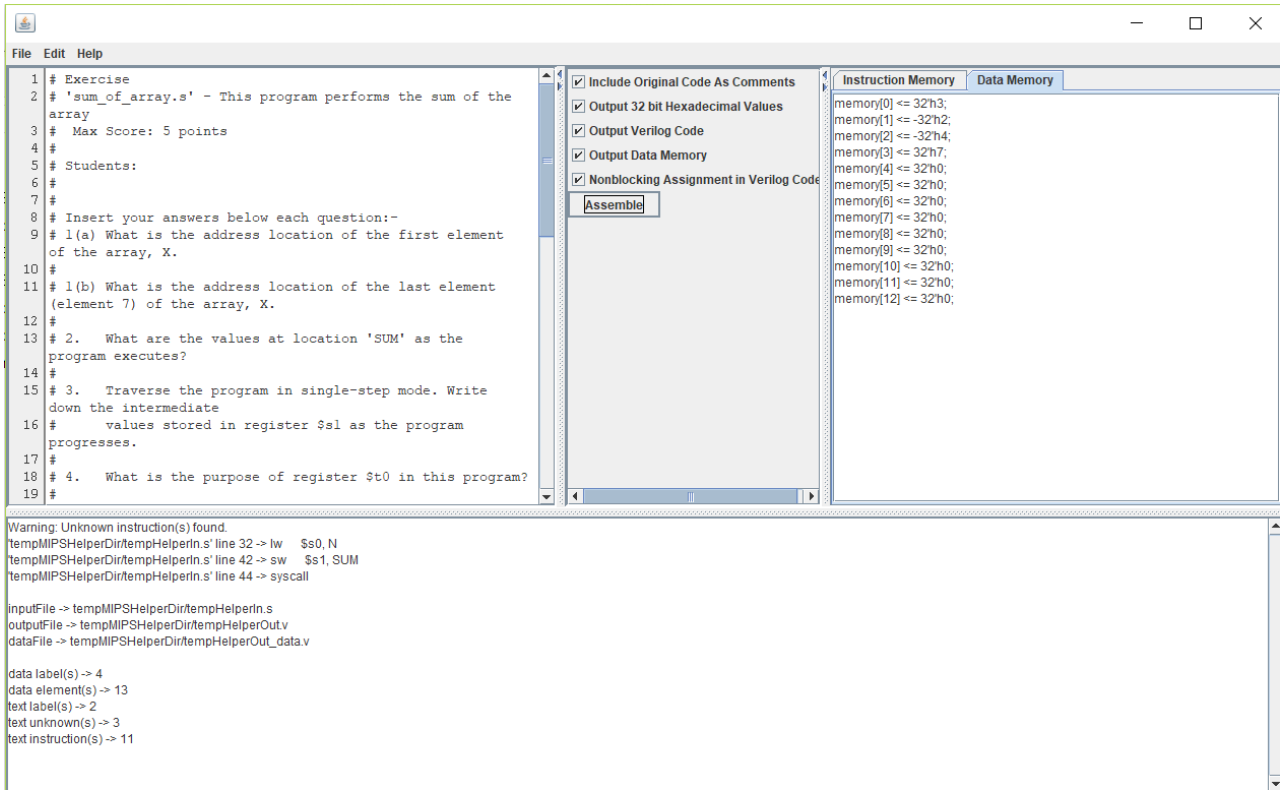
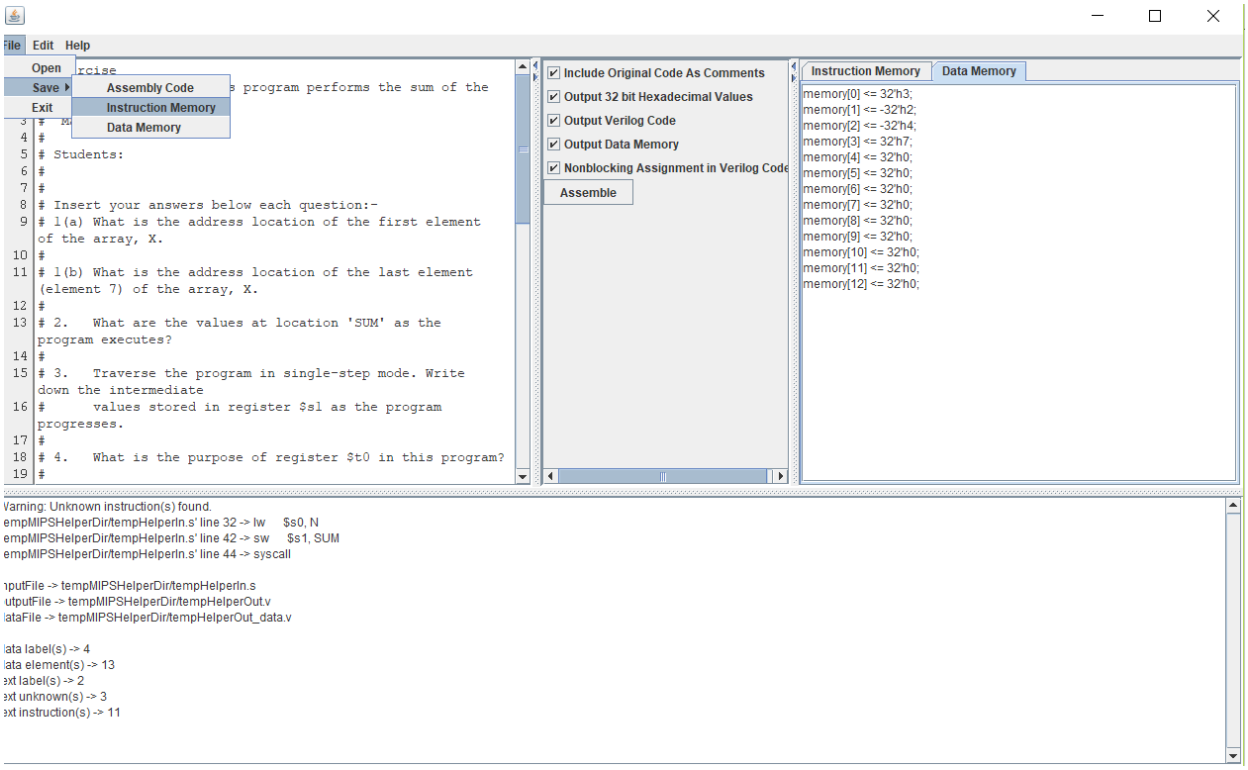


Fig: Generated Data Memory contents after assembling.

(c) After the conversion is done, save the **Instruction Memory** and **Data Memory** contents using the options **File → Save → Instruction Memory** and **File → Save → Data Memory**, respectively. Save the files in desired location (preferably in .txt format).



The saved files can be used to initialized Instruction memory and Data memory.