The University of Newcastle
School of Electrical Engineering and Computing
COMP3290 Compiler Design
Semester 2, 2020

The SM20 Simulator - User Manual

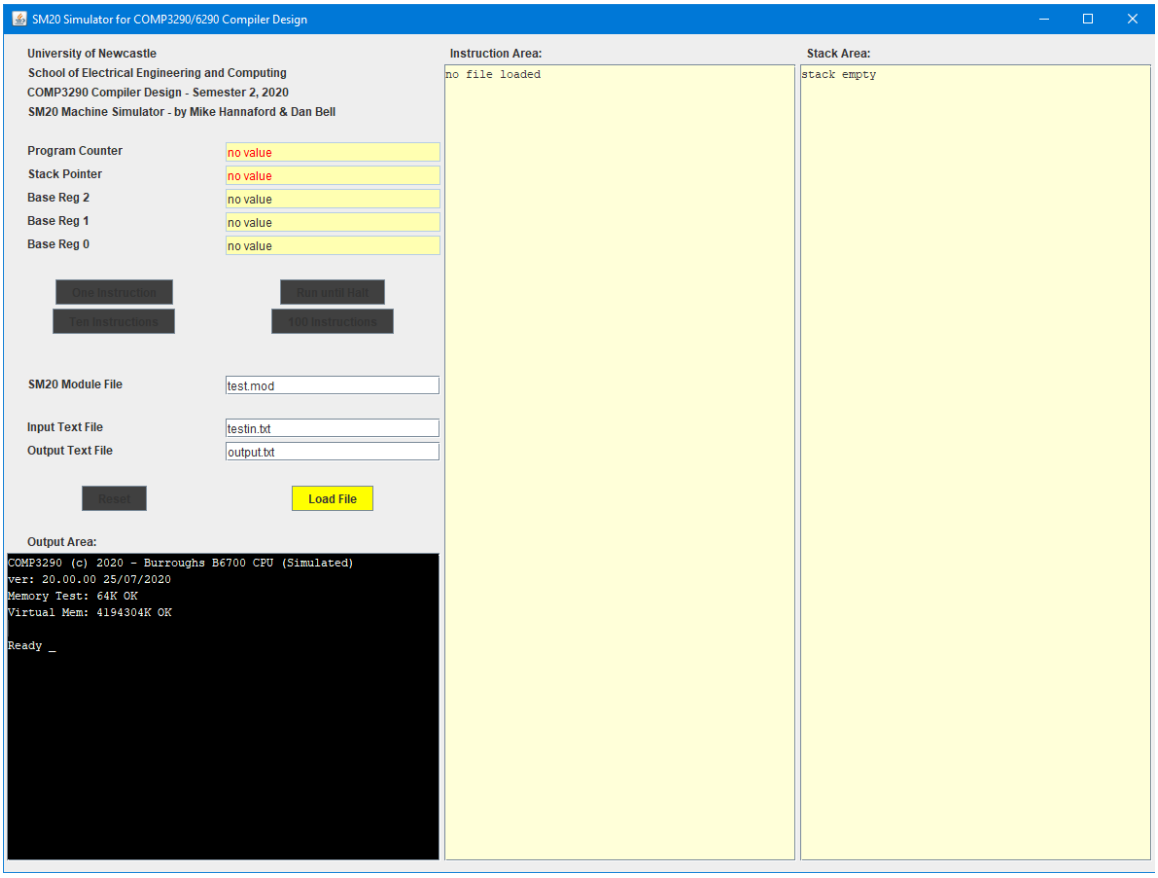
Introduction:

During this semester, the projects in COMP3290 will focus on writing a complete compiler for the language CD20, for the machine known as the SM20 (Stack Machine 2020). Part of this project is to evaluate just how useful a device such as the SM20 will be, by evaluating the complexity and utility of programs that can be executed on it. You will execute your compiled CD20 programs on a Simulator for this architecture.

Running the SM20:

The command **`java -jar SM20.jar`** will start the SM20 machine simulator running. A default empty Machine State Window is displayed. This default display contains default filenames for the *SM20 Module File*, the *Input File*, and the *Output File*. These can be altered or the default values used.

The display uses the javax.swing *GridBagLayout* library and so resizing of the display window in any direction results in the sizes of all window components being re-calculated and re-displayed.



Entering Names of Files:

The simulator works under a Graphical User Interface, which has fields where the user may enter three filenames. All three are simple text files and the GUI does have default names for each, which can then be altered before a button is selected:

SM20 Module Filename

This contains the integer constants, floating point constants, string constants and instructions. All data storage is allocated at run time, on the stack.

Input Filename

This file contains all the data values, in the order that they will be input (when requested via **READI** and **READF** instructions).

Output Filename

This file will be reset to zero length & will await instructions such as **VALPR**, **STRPR**, **CHRPR**, **NEWLN** and **SPACE** to write out the results of a program run. A default output.txt file is included that will record the latest output from the simulator (echoing what is printed to the 'Terminal' in the GUI), to be used for debugging should you wish.

The Buttons and the Display:

The buttons change colour where possible, to indicate to the user which operations are valid at that time. *eg*. when the simulator starts, only the ***"Load File"*** is a valid option.

After a program has been run and a **HALT** instruction encountered, then it is also valid to ***"Reset"*** that program, or to Load another SM20 Module File (or re-load the same one).

- If a button shows as Dark-Grey, then selecting it will give an error.
- If it shows Light-Grey then it is not a useful or preferred operation at that time, but is not likely to give an error.
- If a button is Light-Yellow, then it is a "possibly useful" option (eg. a program can be ***"Reset"*** when execution is part the way through).
- If the button shows Yellow then it is one of the preferred (or recommended) options for the Simulator at that time.

Load File

Read the SM20 Module file and load the instruction (and constant data) area. Open the Input and Output text files, ready for I/O. Initialise all the Base Registers, the Entry Point, Program Counter and Stack Pointer. Display all the Registers, the Instruction Space and the Stack Memory Space of the ready-to-execute program.

Reset

Reset the program after it has been run (or during execution one instruction at a time). Clear the stack memory area and re-open the input/output files. Re-initialise all the Base Registers, the Entry Point, Program Counter and Stack Pointer. Display all the Registers, the Instruction Space and the Stack Memory Space of the ready-to-execute program. If the input/output filenames have changed, then these new files will be opened. If the Module Filename has changed, then an error occurs and the simulator goes back to expecting a new Module File to be loaded.

Run

Continually execute instructions of the program until a **HALT** instruction is executed, or until an SM20 Exception is thrown. If a **HALT** instruction is encountered, display the machine's state (see above). If an exception is thrown, display an error message (no refresh of the display until the error message is acknowledged). Note that selecting ***"Run"*** will execute the program from its current state through until it **HALT**s (or thows an exception). The program is not run from its starting state, unless you have most recently chosen ***"Load File"*** or ***"Reset"***. So it is possible to choose the ***"One Instruction"*** button several times and then to choose ***"Run"*** to complete the program execution.

One Instruction

Execute a single instruction. If successful, update the display. If an exception is thrown then output an error message (the display is not updated).

Ten Instructions

Execute ten single instructions, updating the display after each successfully executed instruction. If a **HALT** instruction is executed, the program terminates normally. If an exception is thrown then output an error message (the display is not updated after that instruction).

100 Instructions

Execute 100 single instructions, updating the display after each successfully executed instruction. If a **HALT** instruction is executed, the program terminates normally. If an exception is thrown then output an error message (the display is not updated after that instruction).

The SM20 display has the register value fields set so that they *cannot be altered*. Other display areas can be "*typed over*", but only the filename fields are input to the program.

Volatile registers (such as **pc** and **sp**) on the display are shown in **Red**. Base Reg 2 (**b2**) is displayed in **Black** at the beginning of an execution and becomes **Red** when a sub-program is called, maintaining the **Red** display for the rest of the execution (*to easily show if your project has successfully made use of the sub-program call/return instructions*).

The ***Instruction Area*** and ***Stack Area*** are displayed one word wide to fit as much information as is possible into these text fields before their automatic scrolling needs to be used.

The *Stack Area* will also display one or two memory words past the value of the Stack Pointer Register.

Finally there is the ***Output Area*** which displays any output from the program, formatted to look like a Terminal. Output is also routed to the specified output file (*by default* `output.txt`)

Fatal SM20 Exceptions:

After a Fatal SM20 Exception has been thrown, the simulator will not necessarily abort, but will be in an indeterminate state. All the buttons (other than ***Load File***) may result in inconsistent actions and an inconsistent information display.

When a fatal exception is thrown, the error message displayed will give information as to where the error occurred and what caused it (remember that the pc register holds the *next instruction byte to be fetched*). On some occasions the ***"Reset"*** button may be able to be used to get the program ready for another run (perhaps using ***"One/Ten/100 Instructions"***) so that the execution can be traced and the problem found and rectified. After an exception has been thrown, the button colour indications outlined above, will no longer hold.

The Structure of the Module File:

The SM20 loader expects the module file to have a very specific structure. The file is a simple text file and contains only integer values. It has FOUR sections:

1. an instruction section
2. an integer constant section
3. a floating point constants section
4. a string constants section.

Each section is preceded by a single value, which gives the number of words of storage that will be occupied by that section within the instruction (**b0**) area of the computer. Even though the values in sections 2 and 5 are byte values (*instructions or characters respectively*), the controlling values give *the number of words*. If any section is of zero length then it consists solely of the respective (*0*) controlling value.

Instruction Section

This consists of a value *n*, followed by *8*n* integers, which are the opcodes and offset bytes for *8*n* bytes of instruction space. Once again it is convention to word-fill this area with **HALT** instructions (*opcode 0*). Note that the first instruction byte is assumed to be the entry point (**ep**) of the program.

Integer Constants Section

This consists of a value *n*, followed by *n* values. These values are allowed to be negative.

Floating Point Constants Section

This consists of a value *n*, followed by *n* values. These values are allowed to be negative and must be in simple fixed point format.

String Constants Section

This consists of a value n, followed by *8*n* integers which are the ascii codes for *8*n* characters. Strings are terminated by a null character, which must be stored as part of the string constant, and each new string does not need to start on a word boundary. It is convention to word-fill this section with null characters (*value 0*). The initial value of **sp** will be the last word of the string constants space because the stack will start off empty.

Reading the Input and Module Files:

Each of these files is a simple text file. When the operating system reads from these files, it does so in each case by means of a StreamTokenizer. Consequently, the format of the files is very open, except that (*as stated previously*) when the controlling value of one of the sections is expected, it MUST be there ready to be input.

DB

v1.0 : *2020-07-28*