emuARM

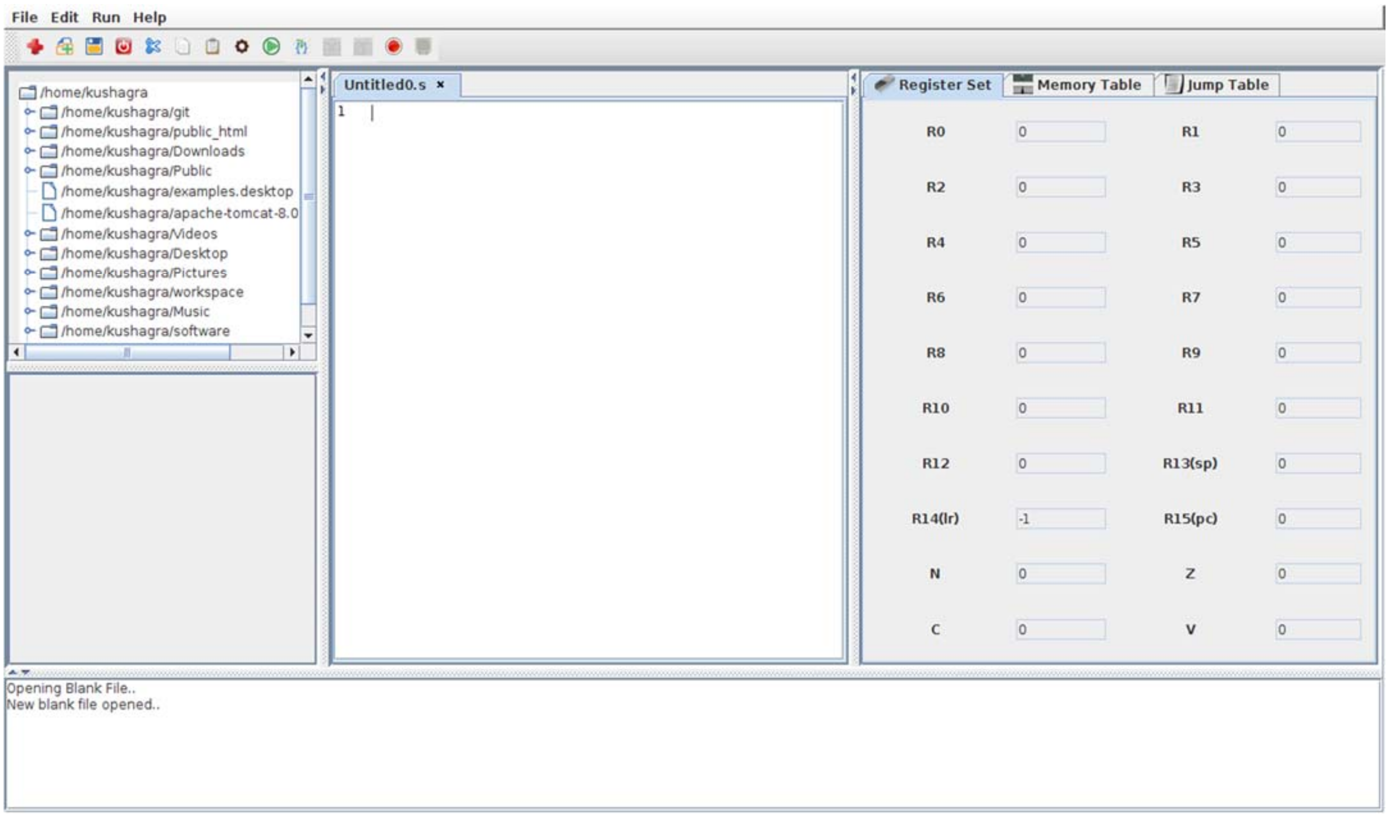
USER MANUAL

LIST OF CONTENTS

1. INTRODUCTION

In today's world, there is an ever increasing array of ARM based mobile devices available in the market, emuARM (Advanced RISC Machine device emulator) is an extremely useful tool available to mobile application developers. The emulator provides a way to test your mobile application on a variety of devices and platforms even if one does not own a physical device. While using one or more real devices can be a valuable aspect of a testing plan, by using the emulator, the developer can verify his application's behaviour across a much wider variety of virtual devices and configurations than would normally be practical using physical devices. Testing an application on the emulator is faster, easier, and more convenient than with physical devices because one can easily reproduce consistent testing scenarios.

The frontend of emuARM is shown below



/home/kushagra
- /home/kushagra/git
- /home/kushagra/public_html
- /home/kushagra/Downloads
- /home/kushagra/Public
- /home/kushagra/examples.desktop
- /home/kushagra/apache-tomcat-8.0
- /home/kushagra/Videos
- /home/kushagra/Desktop
- /home/kushagra/Pictures
- /home/kushagra/workspace
- /home/kushagra/Music
- /home/kushagra/software

R0	0	R1	0
R2	0	R3	0
R4	0	R5	0
R6	0	R7	0
R8	0	R9	0
R10	0	R11	0
R12	0	R13(sp)	0
R14(lr)	-1	R15(pc)	0
N	0	Z	0
C	0	V	0

```
Opening Blank File..
New blank file opened..
```

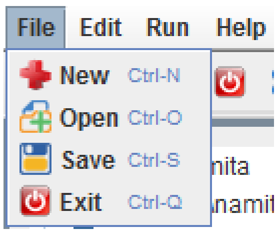
All the operations and components of emuARM are described in the following sections.

1. MENUBAR

The menu bar contains all the options and commands that the user will need in the operation of this emulator from basic file management commands to debugging facilities.

2.1 FILE MENU

This option contains the basic File Management tools for the emulator.

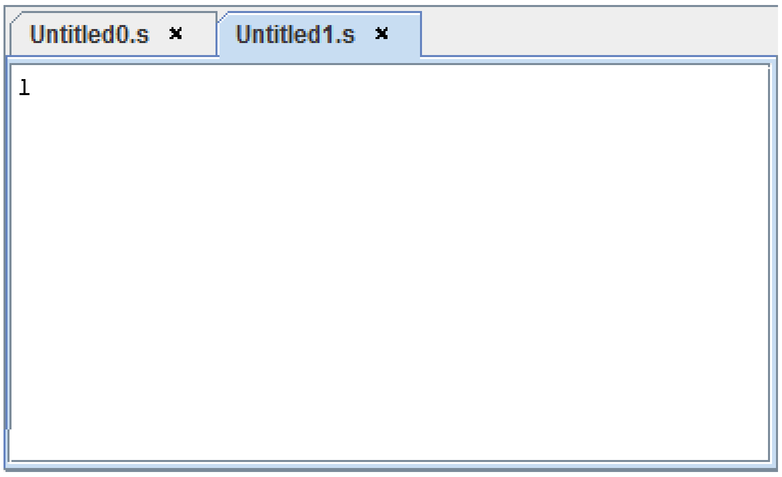


2.1.1 NEW

It is used for creation of a new tab for the text editor where one can enter and edit the code.

It can also be used by the shortcut Ctrl + N.

The new tab will be named Untitledx.s by default, where x is the number of this tab.



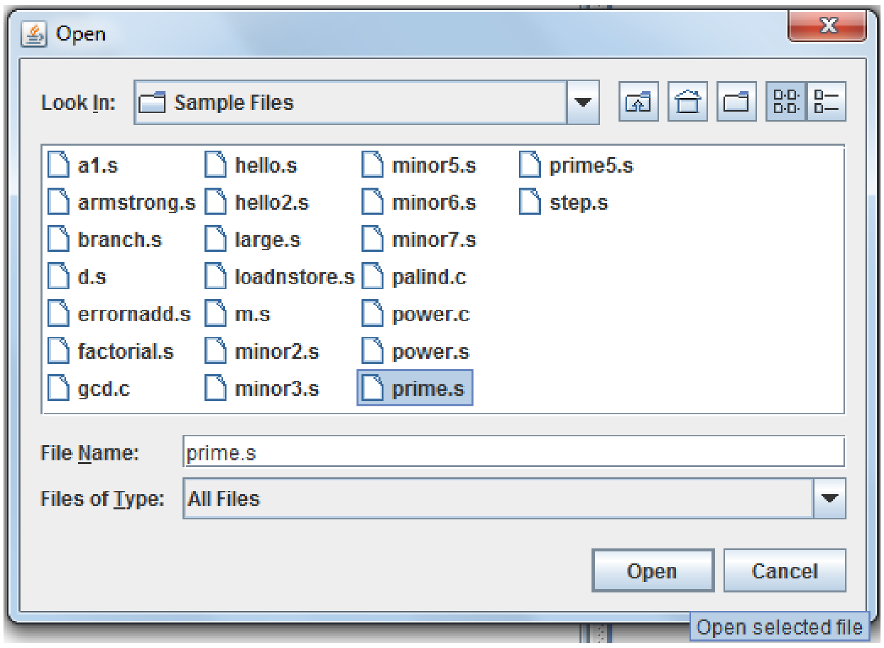
As shown, a new tab opens up by the name of Untitled1.s.

2.1.2 OPEN

Open is used to open an existing file.

After clicking on the Open menu option or pressing Ctrl + O from the keyboard, a dialogue box appears by which we can navigate through the file system and select the necessary file to open.

After clicking on Open, we can now view and edit the required file in the text editor window.



Look In: Sample Files

a1.s, armstrong.s, branch.s, d.s, errornadd.s, factorial.s, gcd.c, hello.s, hello2.s, large.s, loadnstore.s, m.s, minor2.s, minor3.s, minor5.s, minor6.s, minor7.s, palind.c, power.c, power.s, prime.s, prime5.s, step.s

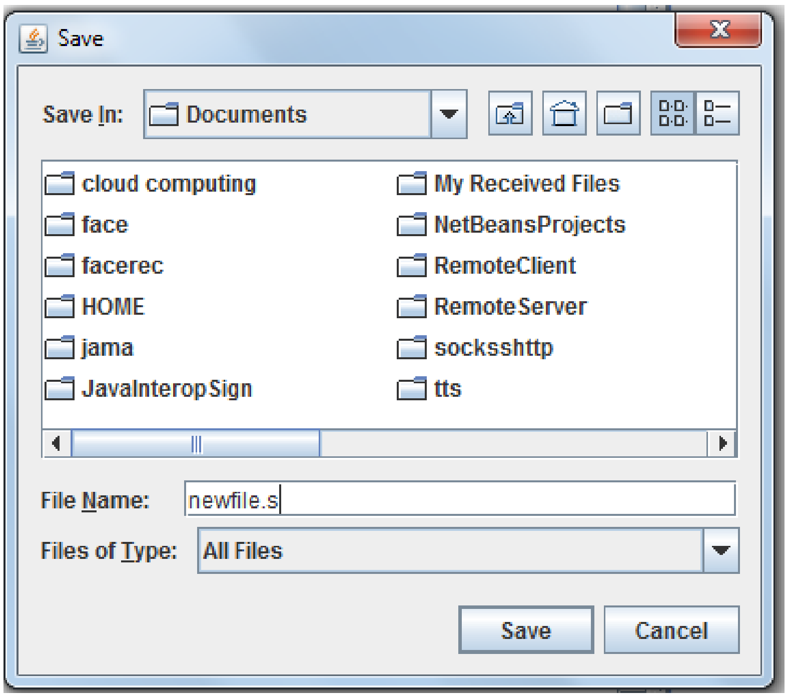
File Name: prime.s

Files of Type: All Files

2.1.3 SAVE

This option can act in two ways. The shortcut for both is Ctrl + S.

If the file to be saved is an Untitled.s file, it acts as a Save As option. A dialog box appears as shown, and the user can enter the desired name and click on Save.



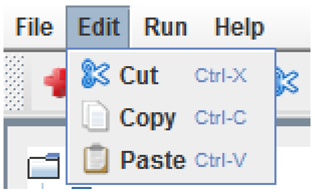
Otherwise, if the file has been saved previously, it acts like a Save button and commits the changes that were made to the file to the memory.

2.1.4 EXIT

On selecting this option or on pressing Ctrl + Q, the user exits from the emulator and the window closes.

2.2 EDIT MENU

This menu has the features for editing the text in files.



2.2.1 CUT

Cut is used to crop the selected portion of the text.

It can be done by selecting the text and then either clicking on Cut menu option or pressing Ctrl +X from the keyboard.

This text can then be placed elsewhere using the Paste option.

2.2.2 COPY

Copy is used to only copy the selected portion of the test and place it in the clipboard to be pasted elsewhere. The original text remains at its place.

Shortcut for this is Ctrl + C.

2.2.3 PASTE

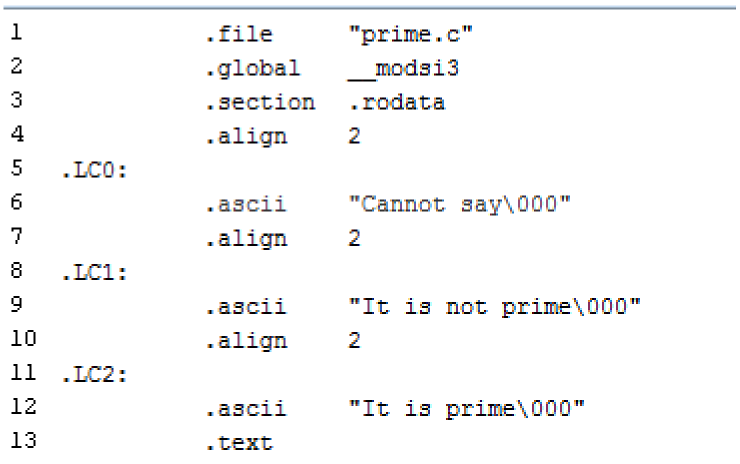
This places the text which was most recently cut or copied to be placed at the location of the cursor.

Shortcut for this option is Ctrl + V.

These three operations are explained below.

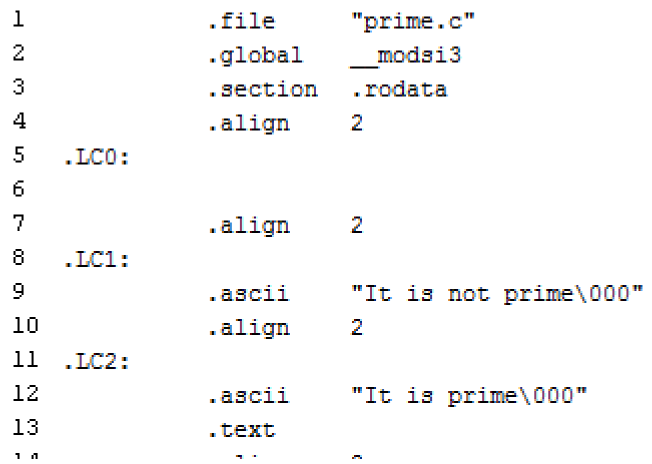
1. Select text and click on Copy.

   The selected text on line 6 is shown in grey colour.



```
1            .file     "prime.c"
2            .global   __modsi3
3            .section  .rodata
4            .align    2
5    .LC0:
6            .ascii    "Cannot say\000"
7            .align    2
8    .LC1:
9            .ascii    "It is not prime\000"
10           .align    2
11   .LC2:
12           .ascii    "It is prime\000"
13           .text
```

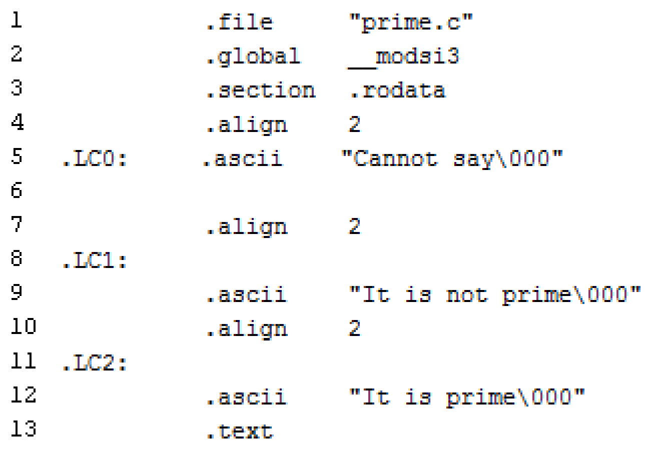
2. Click on Cut if this option is required which would result into the following.



```
1               .file      "prime.c"
2               .global    __modsi3
3               .section   .rodata
4               .align     2
5    .LC0:
6
7               .align     2
8    .LC1:
9               .ascii     "It is not prime\000"
10              .align     2
11   .LC2:
12              .ascii     "It is prime\000"
13              .text
```

Here line 6 has been cropped.

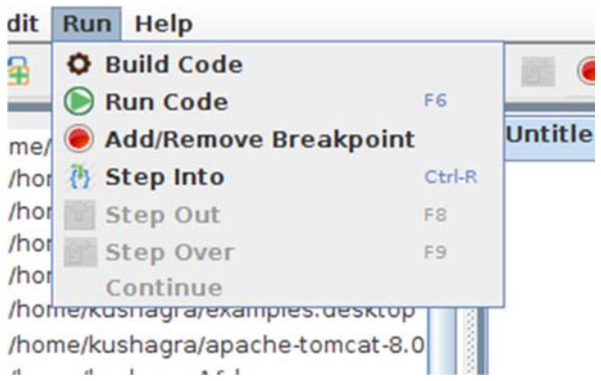
3. Paste is used to place this text on line 5.



```
1               .file      "prime.c"
2               .global    __modsi3
3               .section   .rodata
4               .align     2
5    .LC0:     .ascii     "Cannot say\000"
6
7               .align     2
8    .LC1:
9               .ascii     "It is not prime\000"
10              .align     2
11   .LC2:
12              .ascii     "It is prime\000"
13              .text
```

2.3 RUN MENU

This menu has the options for executing and debugging the programs.



2.3.1 BUILD CODE

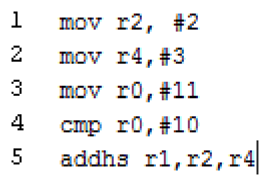
This option is used to compile the whole program. If there are errors in the build process, we get the BUILD FAILED along with the specific error message in the status window. If there are no errors, then we get BUILD SUCCESSFUL in the status window.

2.3.2 RUN CODE

This option is used to execute the whole program and show the output in the status window, reflect the changes in the register set, memory table, jump table, whichever is applicable. If the program has not been built, then it is built first and if there are errors, they are reflected on the status window.

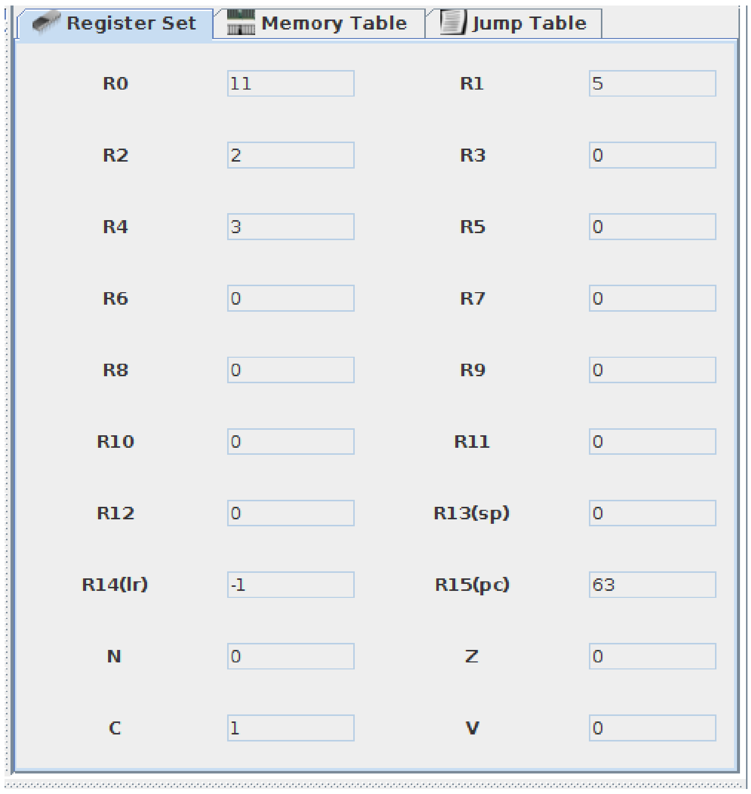
The shortcut for the same is F6.

Example: The following code for conditional execution of the AND instruction is run.



```
mov r2, #2
mov r4,#3
mov r0,#11
cmp r0,#10
addhs r1,r2,r4
```

The register set shows the following values.



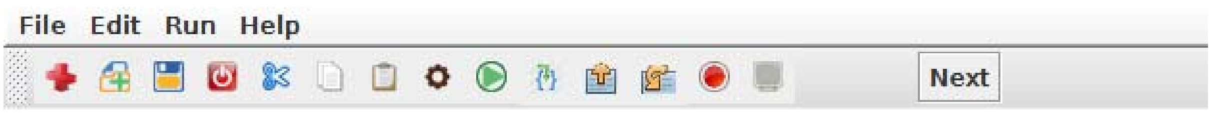
R0	11	R1	5
R2	2	R3	0
R4	3	R5	0
R6	0	R7	0
R8	0	R9	0
R10	0	R11	0
R12	0	R13(sp)	0
R14(lr)	-1	R15(pc)	63
N	0	Z	0
C	1	V	0

2.3.3 STEP INTO

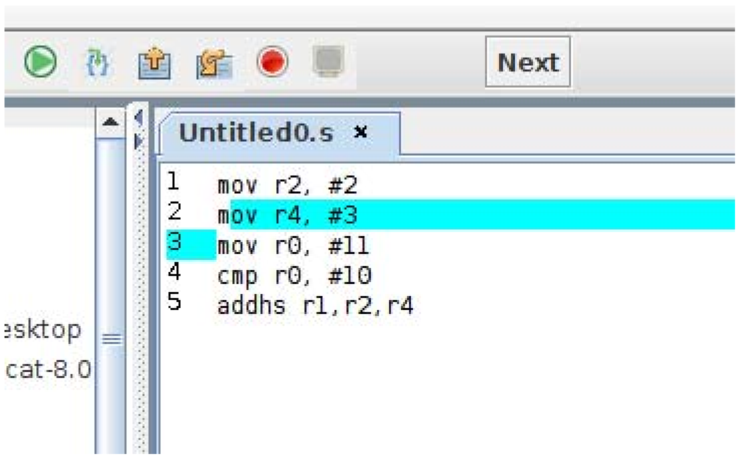
This is a debug mode which executes the code one line at a time sequentially and we can view the results of executing till that line simultaneously in the register set or wherever applicable. The currently executing line is highlighted for ease of reference.

We can also debug only a portion of the code using breakpoints.

After clicking on Step Into, a Next button appears adjacent to the toolbar.

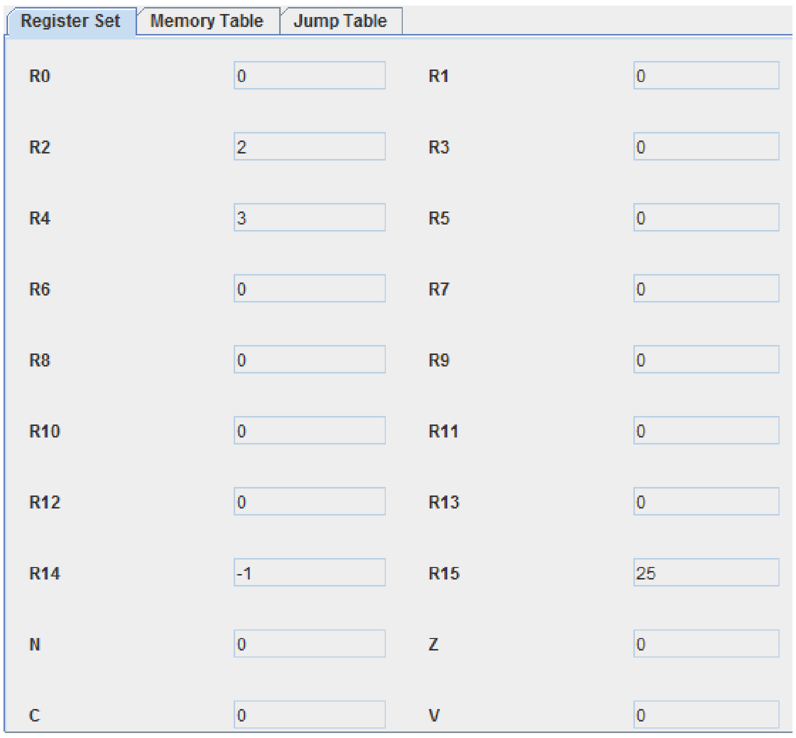


Clicking on this will take the user to the first line to be executed, after which every click will correspond to the execution of the subsequent line.



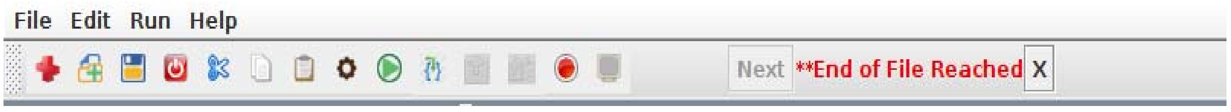
```
1  mov r2, #2
2  mov r4, #3
3  mov r0, #11
4  cmp r0, #10
5  addhs r1,r2,r4
```

Corresponding output till this line (line number 2) is as follows.



R0	0	R1	0
R2	2	R3	0
R4	3	R5	0
R6	0	R7	0
R8	0	R9	0
R10	0	R11	0
R12	0	R13	0
R14	-1	R15	25
N	0	Z	0
C	0	V	0

When the user reaches the last line of the code (or the portion of the code) to be debugged, the Next button and the other debugging modes are deactivated and a warning “End of File reached”, appears to the right of the Next button. There is a cross button next to this warning that will take the user out of the debug mode.



**End of File Reached X

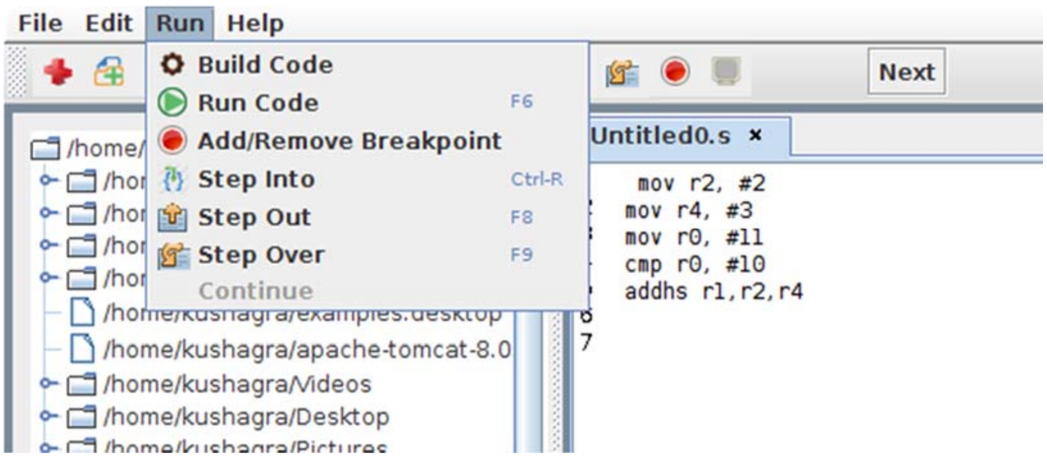
Also, as soon as the Step Into button is clicked, the other debug modes (Step Out, Step Over) get activated. The pane for setting breakpoints also gets activated as shown below.

The operation of these will be explained in the subsequent sections.

The shortcut for this command is Ctrl + R.

2.3.4 STEP OUT

The Step Out debug mode returns from a function which has been stepped into. Even though we return from the function, the remainder of the code inside the method will be executed normally.



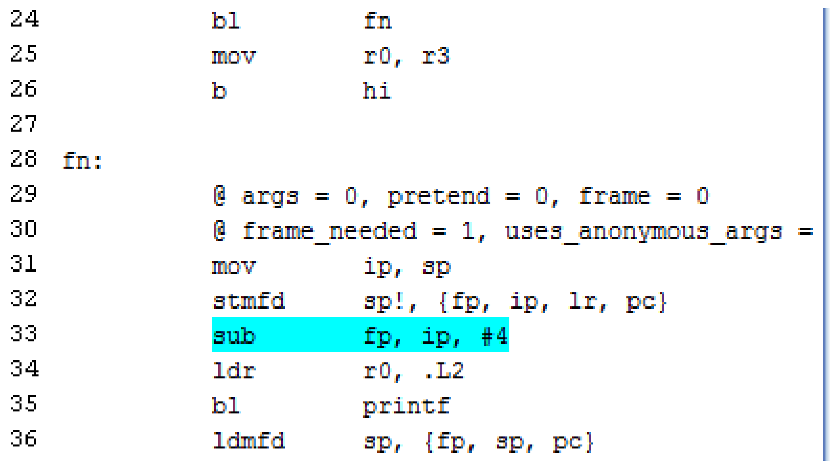
```
 mov r2, #2
mov r4, #3
mov r0, #11
cmp r0, #10
addhs r1,r2,r4
```

Therefore when the user is executing a line within a function and Step Over is clicked, it brings the user to outside the function with only one click of the Next button. The subsequent lines of the code in the function are executed in one go. Also, if this command is invoked when the current line to be debugged is a function call, the definition of the function is executed in one go and the user is taken to the line next to the function call. In all other circumstances it behaves like Step Into.

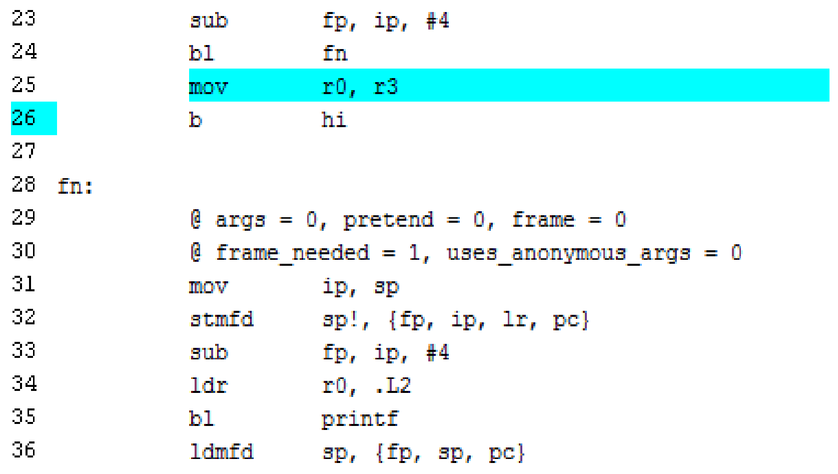
Shortcut for this command is F8.

This mode gets activated only when the program is in debug mode, i.e. Step Into is clicked on.

At line 33, Step Out is called at which point it executes the rest of the lines of the function normally and reaches line 25(the line after the function call) as shown in the second figure.



```
            bl        fn
            mov       r0, r3
            b         hi

fn:
            @ args = 0, pretend = 0, frame = 0
            @ frame_needed = 1, uses_anonymous_args =
            mov       ip, sp
            stmfd     sp!, {fp, ip, lr, pc}
            sub       fp, ip, #4
            ldr       r0, .L2
            bl        printf
            ldmfd     sp, {fp, sp, pc}
```



```
            sub       fp, ip, #4
            bl        fn
            mov       r0, r3
            b         hi

fn:
            @ args = 0, pretend = 0, frame = 0
            @ frame_needed = 1, uses_anonymous_args = 0
            mov       ip, sp
            stmfd     sp!, {fp, ip, lr, pc}
            sub       fp, ip, #4
            ldr       r0, .L2
            bl        printf
            ldmfd     sp, {fp, sp, pc}
```

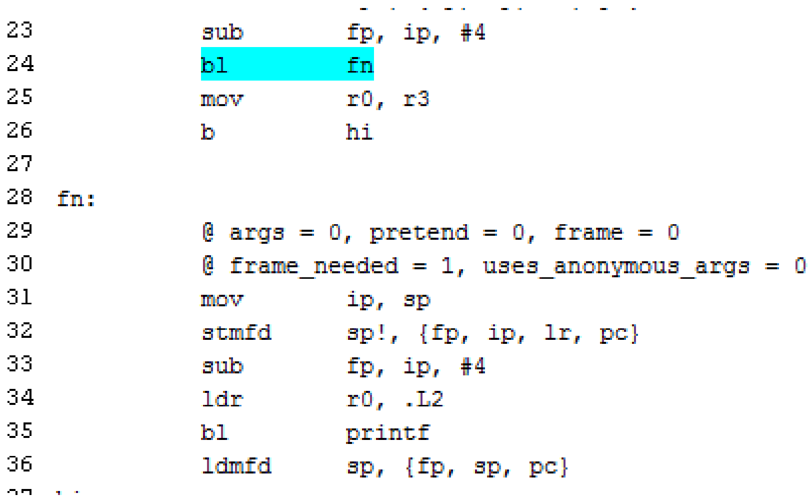
2.3.5 STEP OVER

Step Over command is used for stepping over the next function call (without entering it) at the currently executing line of code. Even though the method is never stepped into, the method will be executed normally.

The shortcut for this is F9.

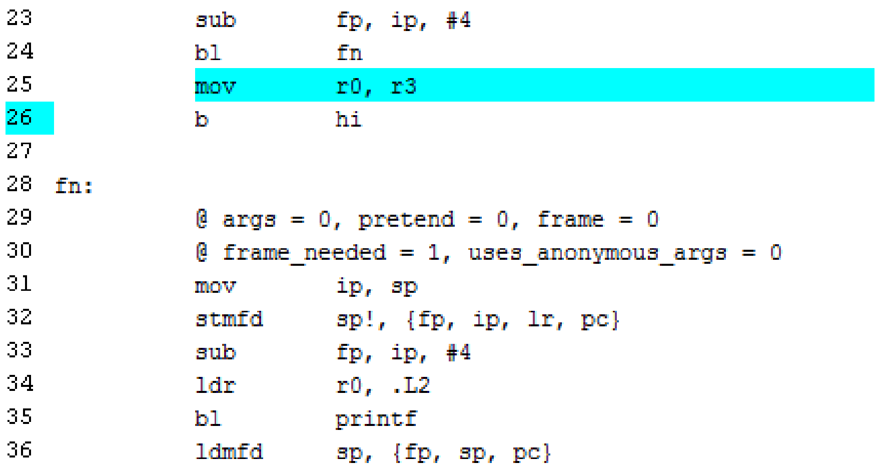
This mode too gets activated only when the program is in debug mode.

The following figures explain the functioning of this debug mode.



```
23           sub      fp, ip, #4
24           bl       fn
25           mov      r0, r3
26           b        hi
27
28  fn:
29           @ args = 0, pretend = 0, frame = 0
30           @ frame_needed = 1, uses_anonymous_args = 0
31           mov      ip, sp
32           stmfd    sp!, {fp, ip, lr, pc}
33           sub      fp, ip, #4
34           ldr      r0, .L2
35           bl       printf
36           ldmfd    sp, {fp, sp, pc}
```

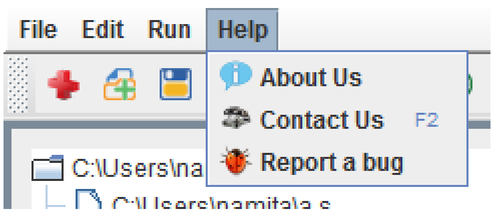
On clicking on Step Over at the function call on line 24, we execute the definition of the function normally and reach line 25 as seen below.



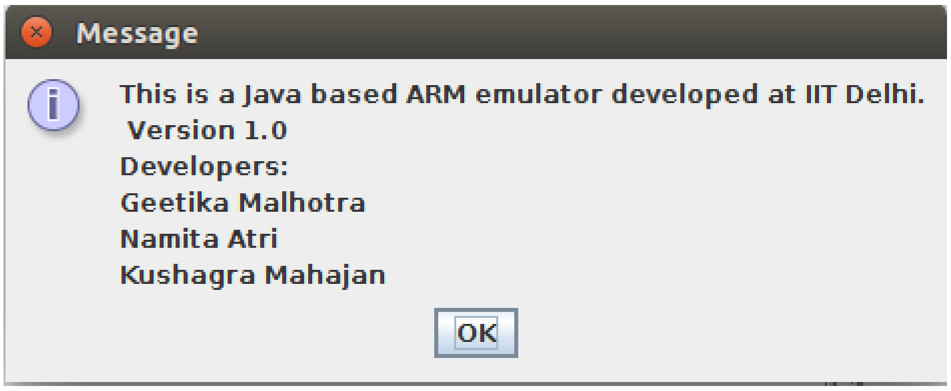
```
23           sub      fp, ip, #4
24           bl       fn
25           mov      r0, r3
26           b        hi
27
28  fn:
29           @ args = 0, pretend = 0, frame = 0
30           @ frame_needed = 1, uses_anonymous_args = 0
31           mov      ip, sp
32           stmfd    sp!, {fp, ip, lr, pc}
33           sub      fp, ip, #4
34           ldr      r0, .L2
35           bl       printf
36           ldmfd    sp, {fp, sp, pc}
```

2.4 HELP MENU

The help menu lists general options for information about the emulator and the developer, and contact details.



2.4.1 ABOUT US

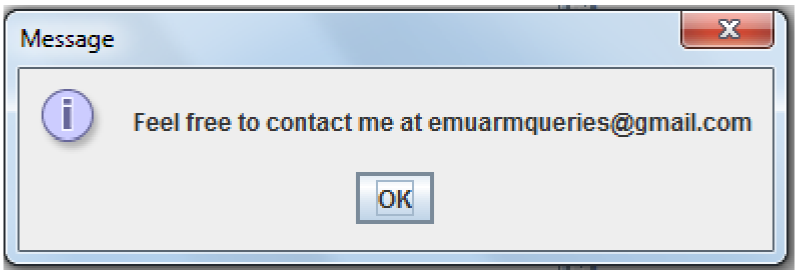

Message

This is a Java based ARM emulator developed at IIT Delhi.
Version 1.0
Developers:
Geetika Malhotra
Namita Atri
Kushagra Mahajan

This displays basic details about the emulator as shown below.

2.4.2 CONTACT US

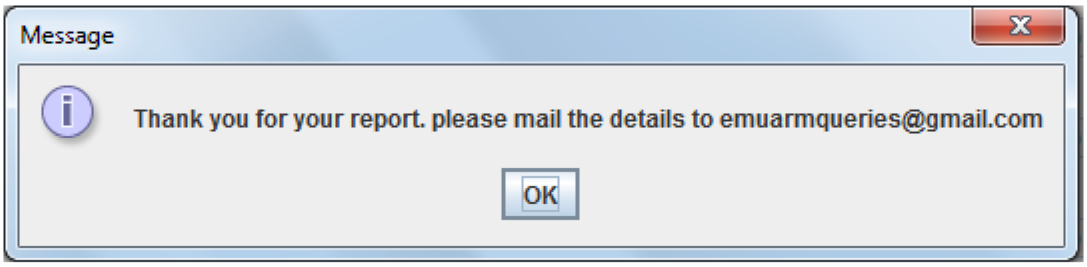
The user can contact the developer for any queries at emuarmqueries@gmail.com.


Message

Feel free to contact me at emuarmqueries@gmail.com

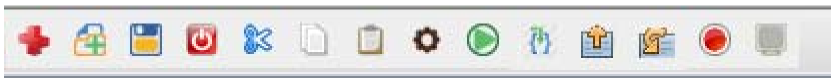
2.4.3 REPORT A BUG

In case of any errors in the emulator, the user can contact the developers of this emulator at emuarmqueries@gmail.com.


Message

Thank you for your report. please mail the details to emuarmqueries@gmail.com

3. TOOLBAR



The toolbar contains shortcuts for various commands that are available in the menu bar.

The icons represent the following actions.

New	Open	Save
Exit	Cut	Copy
Build Code	Paste	Run Code
Step Into	Step Out	Step Over
Breakpoint	Clear the Window	

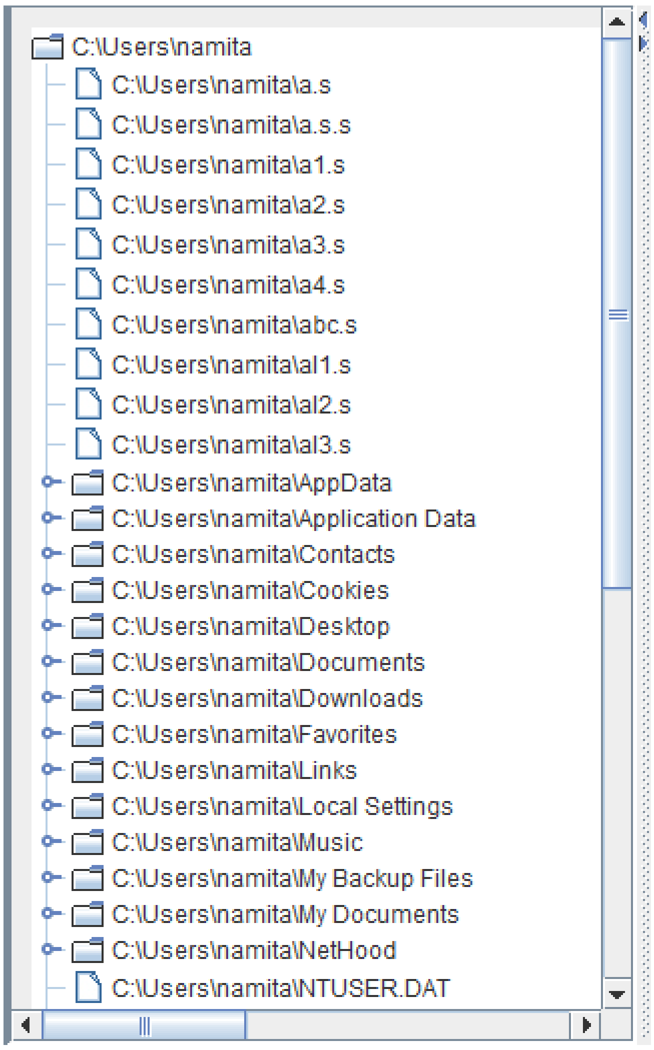
The operation of these has been explained in the previous sections.

4. TREE VIEW DISPLAY

This panel appears on the left side of the emulator and displays the file structure of the computer in a hierarchical format. The root of the tree view points to the home directory. Any existing file can be opened in the text editor through this tree view.

A scroll bar also appears for navigation through this tree view.

Double-clicking on a folder will display its structure of files and folder while single-clicking on a file will open it in the text editor.


C:\Users\namita
C:\Users\namita\a.s
C:\Users\namita\a.s.s
C:\Users\namita\a1.s
C:\Users\namita\a2.s
C:\Users\namita\a3.s
C:\Users\namita\a4.s
C:\Users\namita\abc.s
C:\Users\namita\al1.s
C:\Users\namita\al2.s
C:\Users\namita\al3.s
C:\Users\namita\AppData
C:\Users\namita\Application Data
C:\Users\namita\Contacts
C:\Users\namita\Cookies
C:\Users\namita\Desktop
C:\Users\namita\Documents
C:\Users\namita\Downloads
C:\Users\namita\Favorites
C:\Users\namita\Links
C:\Users\namita\Local Settings
C:\Users\namita\Music
C:\Users\namita\My Backup Files
C:\Users\namita\My Documents
C:\Users\namita\NetHood
C:\Users\namita\NTUSER.DAT

5. BREAKPOINTS

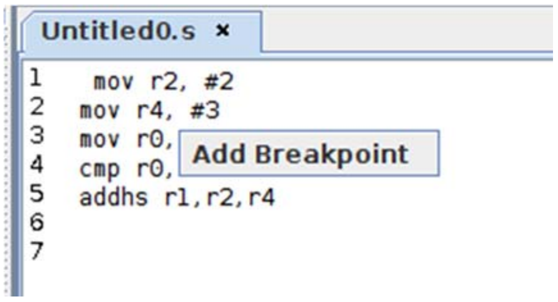
Breakpoints are used for limiting the code to be debugged to a specific portion and can be set through the breakpoint panel.

To add or remove a breakpoint, the user must place the cursor on the line and then either right click or click on the Add/Remove Breakpoint button on the panel. If the line does not have a breakpoint, then the Add Breakpoint option will be displayed else the Remove Breakpoint option will be displayed on right click.

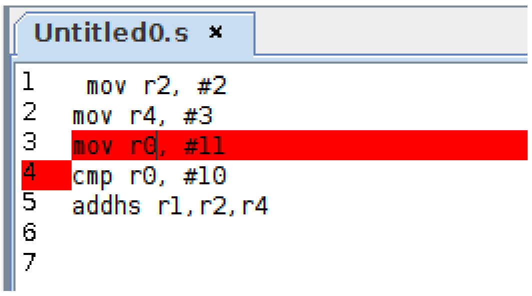
The user can set any number of breakpoints. After setting the breakpoints if we run the code, the code is executed till the line before the breakpoint.

After the first run, the 'Continue' option gets enabled and the user can click that option to run till the next breakpoint and this process can be repeated.

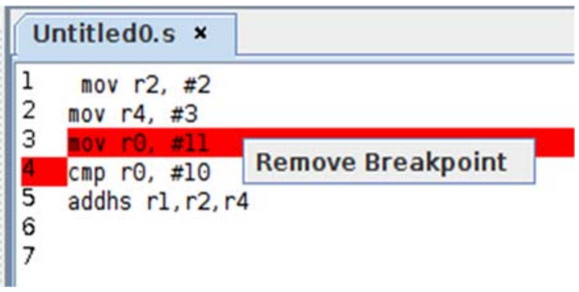
The following sequence shows the operation of breakpoints.



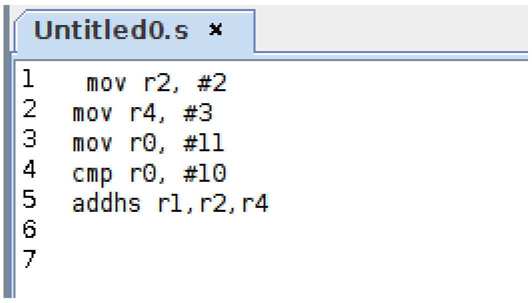
```
 mov r2, #2
mov r4, #3
mov r0,
cmp r0,
addhs r1,r2,r4
```



```
 mov r2, #2
mov r4, #3
mov r0, #11
cmp r0, #10
addhs r1,r2,r4
```



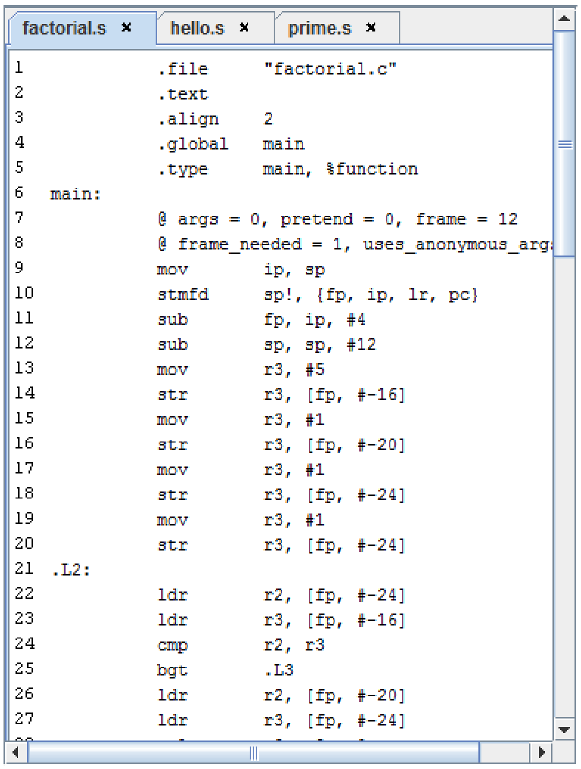
```
 mov r2, #2
mov r4, #3
mov r0, #11
cmp r0, #10
addhs r1,r2,r4
```



```
 mov r2, #2
mov r4, #3
mov r0, #11
cmp r0, #10
addhs r1,r2,r4
```

6. TEXT EDITOR

This is the place where the user enters the program or views an existing program. Multiple tabs can be opened and closed. Line numbers are also displayed.



```
                .file       "factorial.c"
                .text
                .align      2
                .global     main
                .type       main, %function
main:
                @ args = 0, pretend = 0, frame = 12
                @ frame_needed = 1, uses_anonymous_arg
                mov         ip, sp
                stmfd       sp!, {fp, ip, lr, pc}
                sub         fp, ip, #4
                sub         sp, sp, #12
                mov         r3, #5
                str         r3, [fp, #-16]
                mov         r3, #1
                str         r3, [fp, #-20]
                mov         r3, #1
                str         r3, [fp, #-24]
                mov         r3, #1
                str         r3, [fp, #-24]
.L2:
                ldr         r2, [fp, #-24]
                ldr         r3, [fp, #-16]
                cmp         r2, r3
                bgt         .L3
                ldr         r2, [fp, #-20]
                ldr         r3, [fp, #-24]
```

7. REGISTER SET

This displays the values in the register set which contains the following 32 bit registers.

- r0-r12 general purpose registers
- r13 (the stack pointer) and r14 (link register)
- r15 (the program counter)
- cpsr (current program status register)
- spsr (saved program status register)

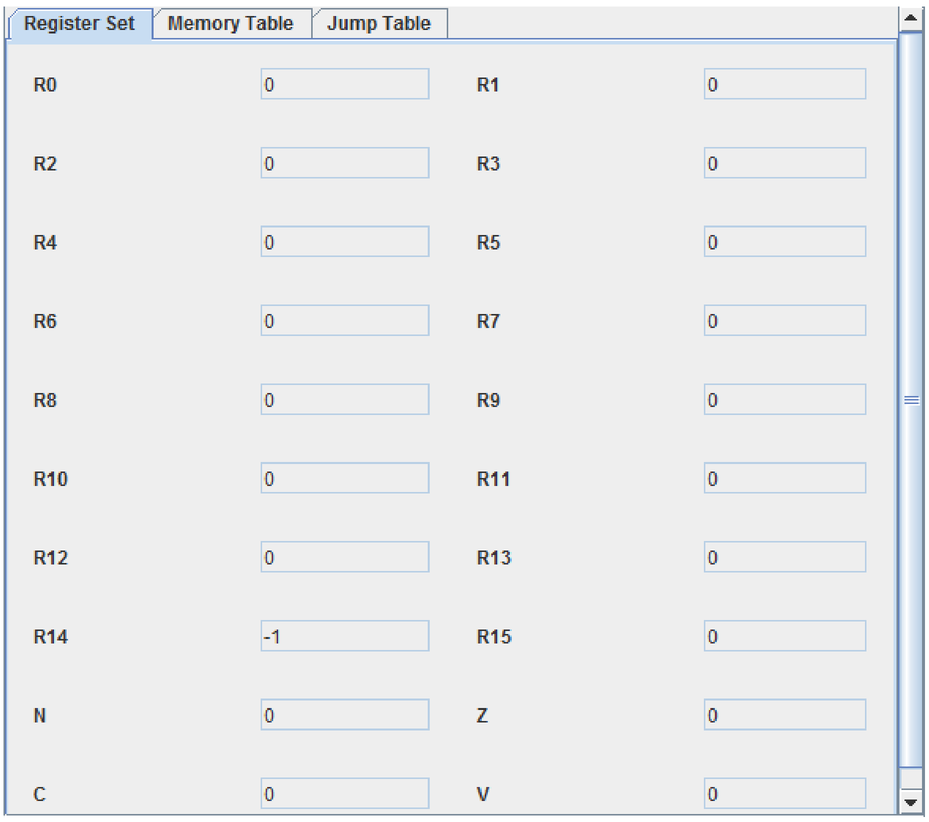
The program status registers have the following condition codes.

N = **N**egative result from ALU flag.

Z = **Z**ero result from ALU flag.

C = ALU operation **C**arried out

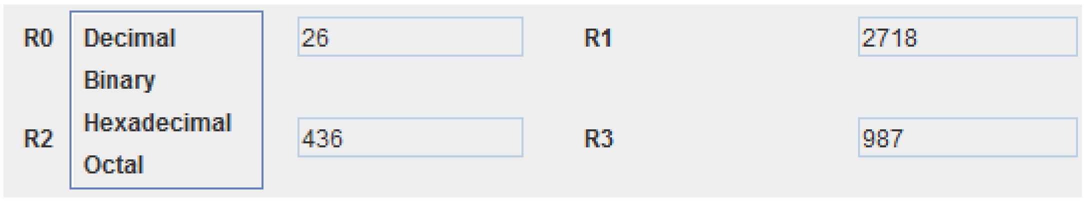
V = ALU operation o**V**erflowed



R0	0	R1	0
R2	0	R3	0
R4	0	R5	0
R6	0	R7	0
R8	0	R9	0
R10	0	R11	0
R12	0	R13	0
R14	-1	R15	0
N	0	Z	0
C	0	V	0

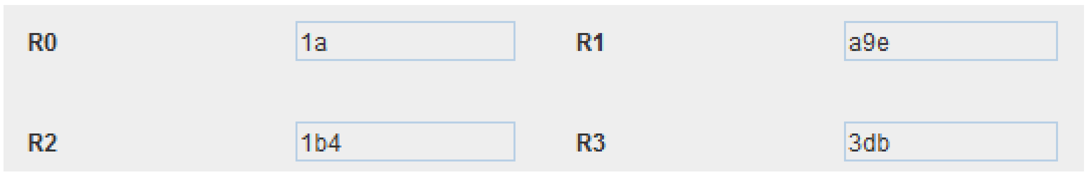
The values in the register set are displayed in the decimal format by default. By right clicking anywhere on the register set, we can set the format to decimal, hexadecimal, binary and octal.

For example, these following values are shown in decimal format (default). We right click and select required format.



R0	26	R1	2718
R2	436	R3	987

The following figure shows the values in hexadecimal representation.



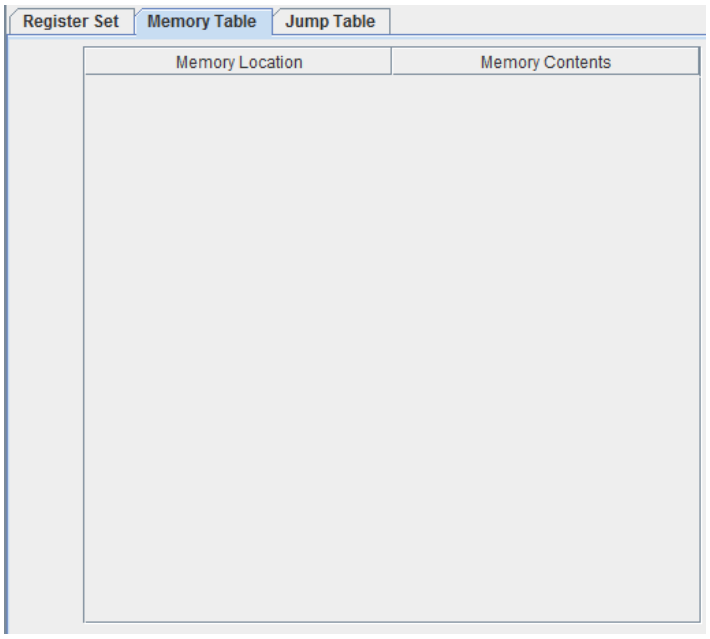
R0	1a	R1	a9e
R2	1b4	R3	3db

8. MEMORY TABLE

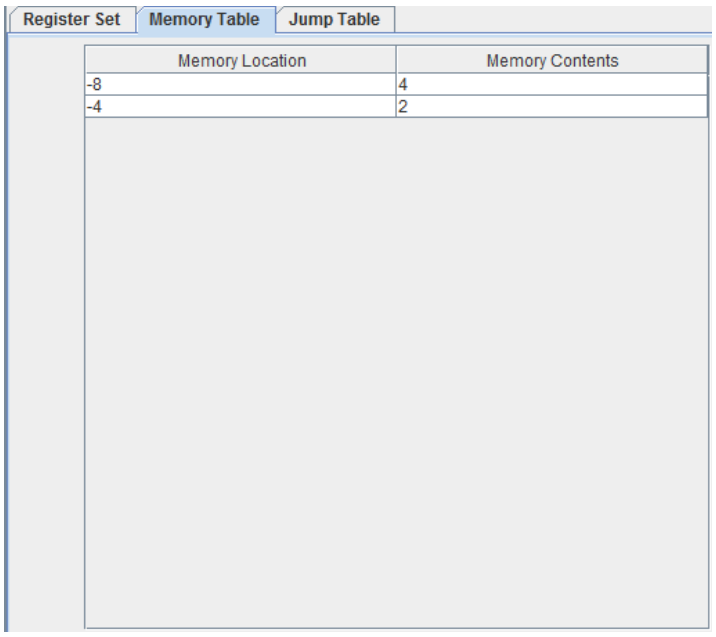
This table displays all the locations at which values are stored in the memory and the respective contents.

Initially the table is empty and progressively augments as a memory location with its respective memory content is added.

These two cases are shown in the following figures.



Memory Location	Memory Contents

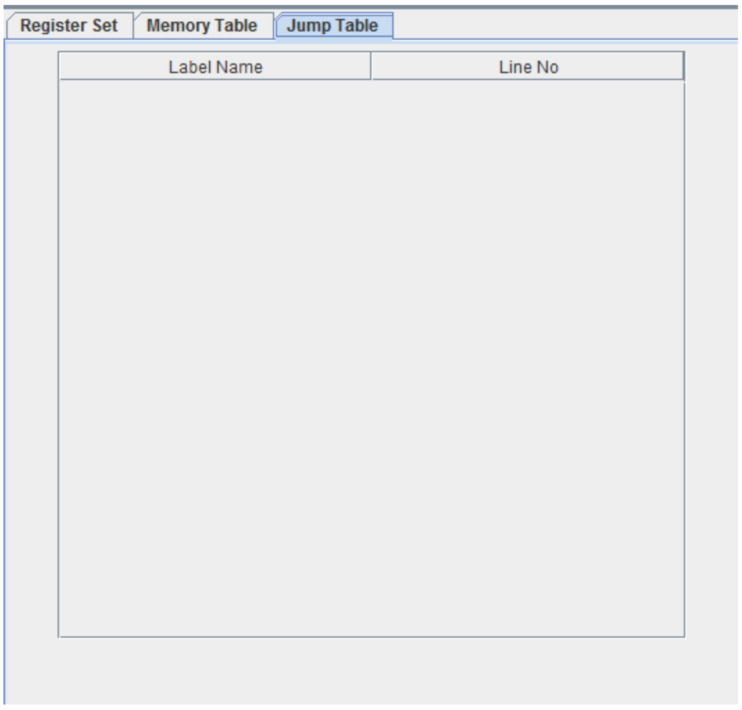


Memory Location	Memory Contents
-8	4
-4	2

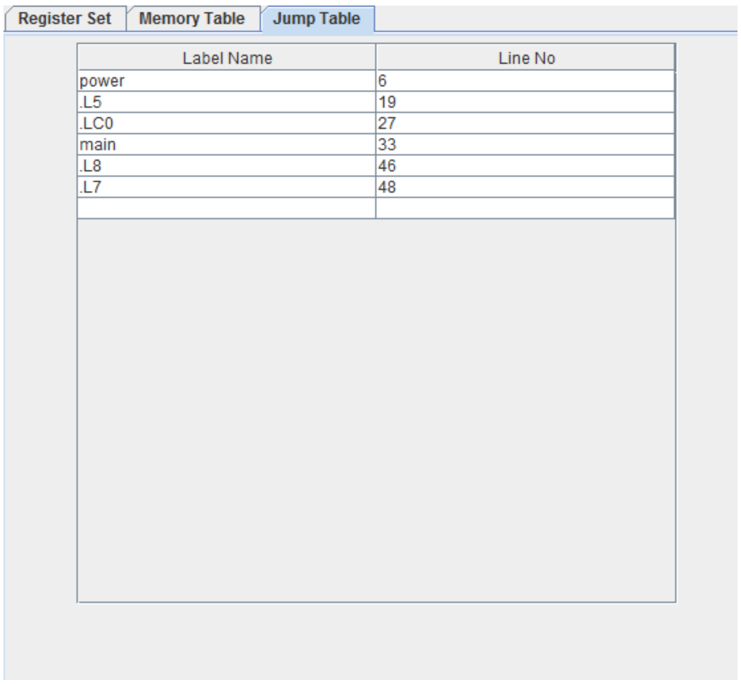
9. JUMP TABLE

This table refers to the one which stores all the labels and the respective line numbers occurring in the program. It is the third tab in the right side panel of the emulator.

Again as with the memory table, the jump table is initially empty, as shown below.



This is filled at the time of compilation and required number of rows are filled up and added to the table. This is shown in the following figure.



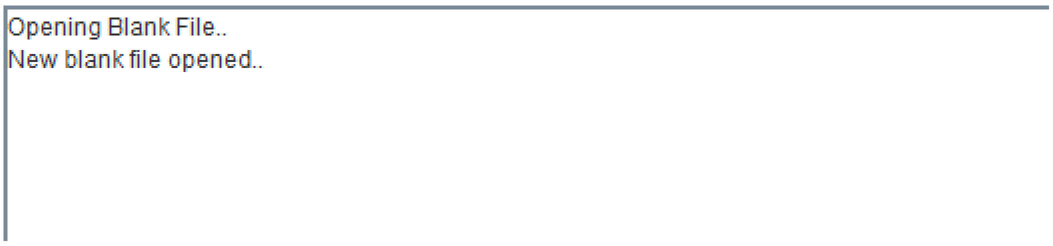
Register Set | Memory Table | Jump Table

Label Name	Line No
power	6
.L5	19
.LC0	27
main	33
.L8	46
.L7	48

10. STATUS WINDOW

This is the window which displays the statuses after completion of various actions.

The status window when the emulator is started is as shown below and corresponds to the opening of a new untitled file in the text editor.

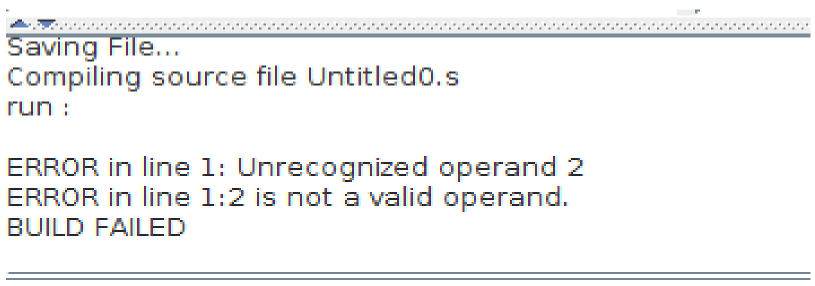


```
Opening Blank File..
New blank file opened..
```

Various such statuses for saving a program, building a program etc. are displayed here. In case of errors of various kinds in entering the program, the respective status describing the status is displayed.

Example: The following statement is erroneous as the # is missing at the front of the operand.

mov r3, 2

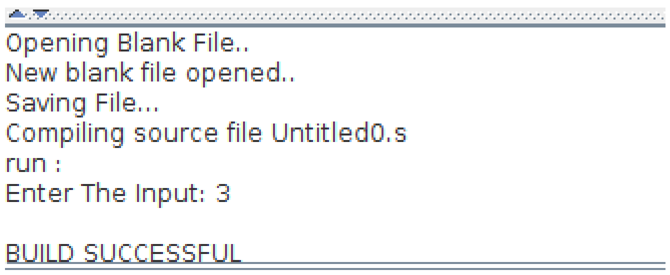

```
Saving File...
Compiling source file Untitled0.s
run :

ERROR in line 1: Unrecognized operand 2
ERROR in line 1:2 is not a valid operand.
BUILD FAILED
```

The status window is also where the user can enter input and the output is printed.

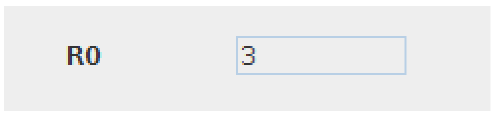
Example: For entering value into register R0, the user can write the following line

.read r0

On executing this line, the program asks the user to enter the input value. After the value, Enter key is pressed.


```
Opening Blank File..
New blank file opened..
Saving File...
Compiling source file Untitled0.s
run :
Enter The Input: 3

BUILD SUCCESSFUL
```

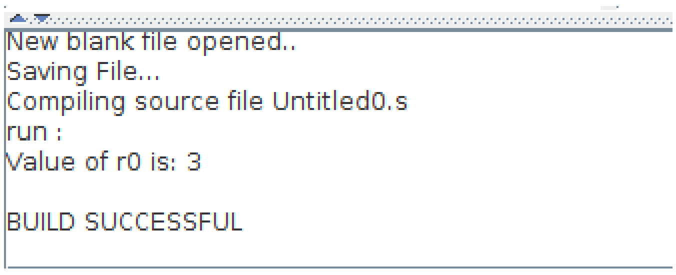
The new value is reflected in the required register.


```
R0        3
```

Just like read, register values can also be printed using the .print command.

Example: For printing the value of register R0, the user can write the following line

.print r0



```
New blank file opened..
Saving File...
Compiling source file Untitled0.s
run :
Value of r0 is: 3

BUILD SUCCESSFUL
```