

COMP2611: Computer Organization

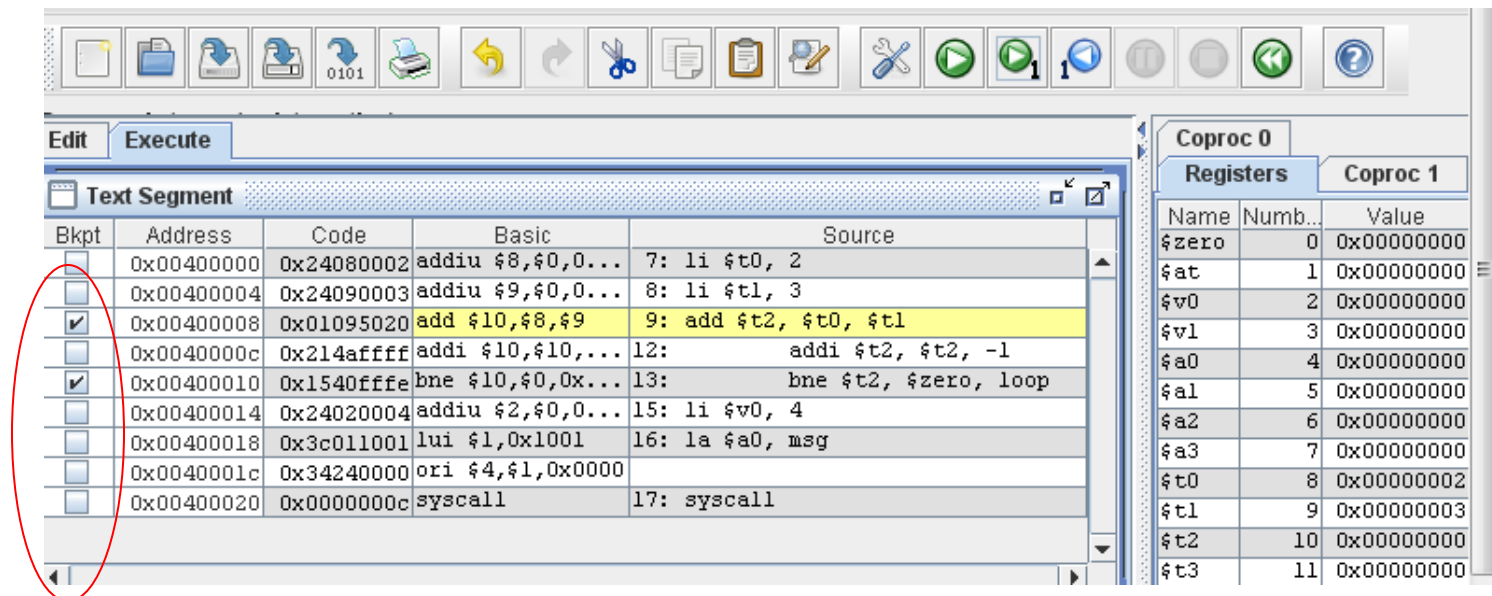
Debugging on MARS

Overview

- You will learn the following in this lab:
 - how to debug a MIPS program using the debugging features in MARS.

Tracing program execution

- MARS provides many debugging features:
 - Breakpoint -- pauses the execution at an instruction.
 - Click the box on the column "Bkpt" of an instruction to enable a breakpoint there.
 - All the instructions before the breakpoint were executed, and all the subsequent instructions, including the one at the breakpoint, are not executed yet.



MARS Breakpoint

- Load the example program `debug1.s`, and try enabling the breakpoints shown below.
- Then start the execution to see how it pauses at the first breakpoint.
- The values of the registers and memory reflect the execution up to (but not including) the instruction at this breakpoint.

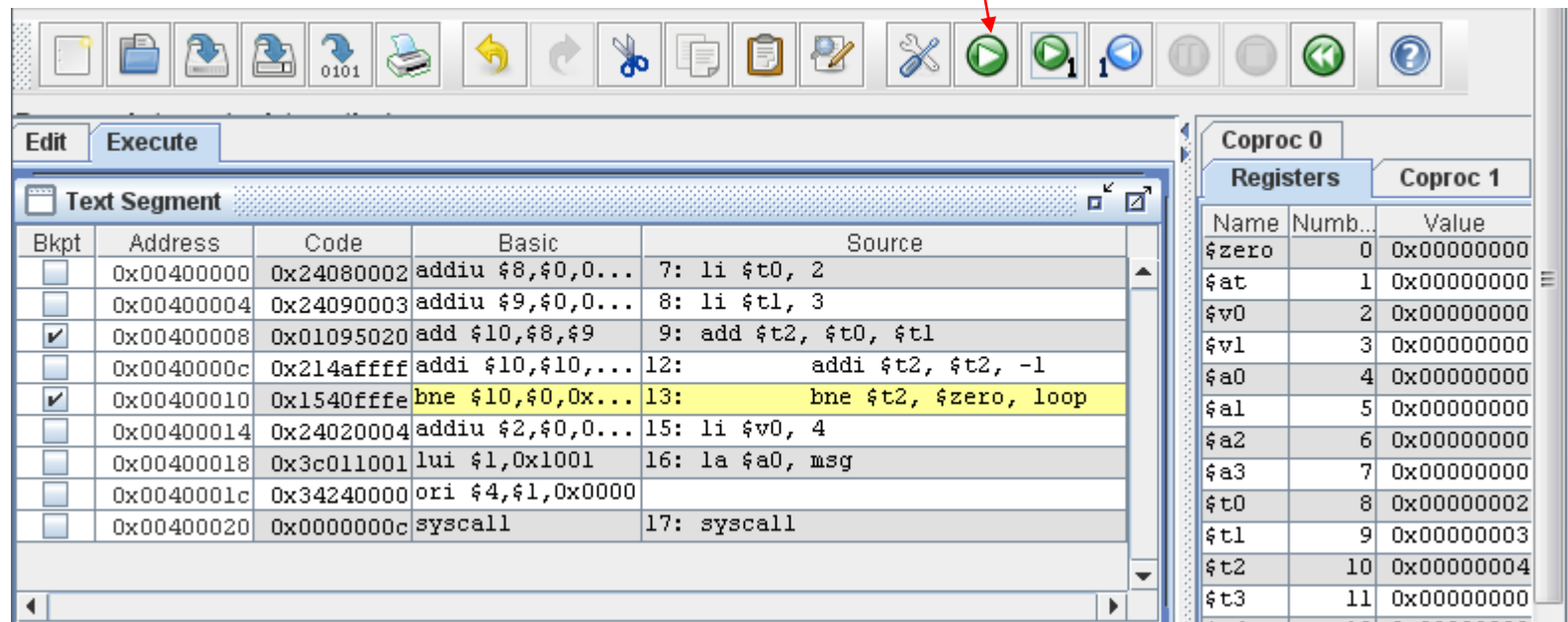
The screenshot shows the MARS MIPS simulator interface. The top toolbar contains various icons for file operations, execution, and debugging. Below the toolbar, the 'Edit' and 'Execute' tabs are visible. The 'Text Segment' window displays a list of instructions with their addresses, codes, basic forms, and source code. Two breakpoints are enabled, indicated by checked boxes in the 'Bkpt' column. The first breakpoint is at address 0x00400008, and the second is at address 0x00400010. The instruction at 0x00400008 is highlighted in yellow. The 'Registers' window on the right shows the values of the MIPS registers, including \$zero, \$at, \$v0, \$v1, \$a0, \$a1, \$a2, \$a3, \$t0, \$t1, \$t2, and \$t3.

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x24080002	addiu \$8,\$0,0...	7: li \$t0, 2
<input type="checkbox"/>	0x00400004	0x24090003	addiu \$9,\$0,0...	8: li \$t1, 3
<input checked="" type="checkbox"/>	0x00400008	0x01095020	add \$10,\$8,\$9	9: add \$t2, \$t0, \$t1
<input type="checkbox"/>	0x0040000c	0x214affff	addi \$10,\$10,...	12: addi \$t2, \$t2, -1
<input checked="" type="checkbox"/>	0x00400010	0x1540fffe	bne \$10,\$0,0x...	13: bne \$t2, \$zero, loop
<input type="checkbox"/>	0x00400014	0x24020004	addiu \$2,\$0,0...	15: li \$v0, 4
<input type="checkbox"/>	0x00400018	0x3c011001	lui \$1,0x1001	16: la \$a0, msg
<input type="checkbox"/>	0x0040001c	0x34240000	ori \$4,\$1,0x0000	
<input type="checkbox"/>	0x00400020	0x0000000c	syscall	17: syscall

Coproc 0		
Registers		
Name	Numb...	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000002
\$t1	9	0x00000003
\$t2	10	0x00000000
\$t3	11	0x00000000

MARS Breakpoint

- Start the execution again (click the same button) to see how the execution pauses at the second breakpoint.



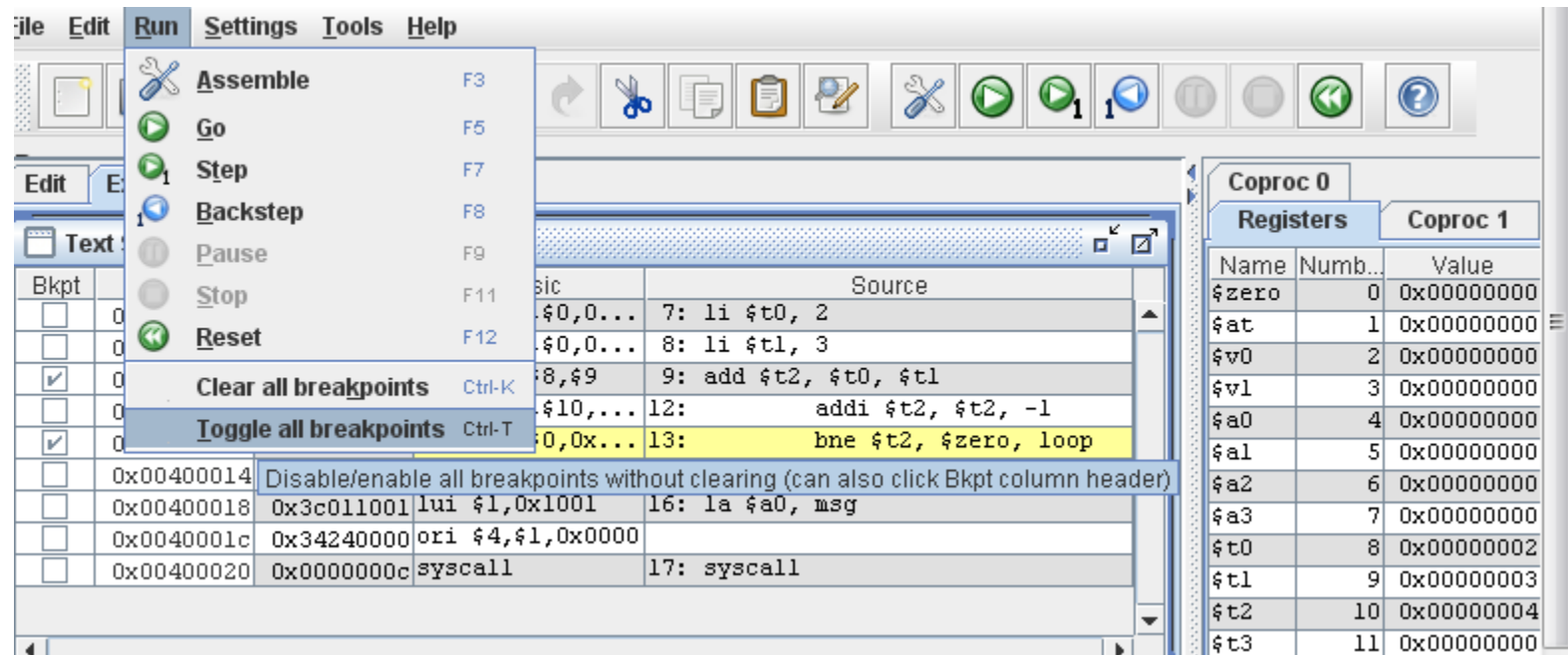
The screenshot shows the MARS IDE interface. At the top, a toolbar contains various icons for file operations and execution. A red arrow points to the 'Run' button (a green play icon) in the toolbar. Below the toolbar, the 'Execute' tab is active, displaying a 'Text Segment' table with columns for Bkpt, Address, Code, Basic, and Source. The table lists assembly instructions with their addresses and basic blocks. The instruction at address 0x00400010, 'bne \$t0, \$zero, loop', is highlighted in yellow and has a checked breakpoint. To the right of the text segment, the 'Registers' panel shows the state of Coprocessor 0 registers, including \$zero, \$at, \$v0, \$v1, \$a0, \$a1, \$a2, \$a3, \$t0, \$t1, \$t2, and \$t3, along with their numerical values.

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x24080002	addiu \$8,\$0,0...	7: li \$t0, 2
<input type="checkbox"/>	0x00400004	0x24090003	addiu \$9,\$0,0...	8: li \$t1, 3
<input checked="" type="checkbox"/>	0x00400008	0x01095020	add \$10,\$8,\$9	9: add \$t2, \$t0, \$t1
<input type="checkbox"/>	0x0040000c	0x214affff	addi \$10,\$10,...	12: addi \$t2, \$t2, -1
<input checked="" type="checkbox"/>	0x00400010	0x1540fffe	bne \$t0,\$0,0x...	13: bne \$t2, \$zero, loop
<input type="checkbox"/>	0x00400014	0x24020004	addiu \$2,\$0,0...	15: li \$v0, 4
<input type="checkbox"/>	0x00400018	0x3c011001	lui \$1,0x1001	16: la \$a0, msg
<input type="checkbox"/>	0x0040001c	0x34240000	ori \$4,\$1,0x0000	
<input type="checkbox"/>	0x00400020	0x0000000c	syscall	17: syscall

Coproc 0		
Registers		
Name	Numb...	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000002
\$t1	9	0x00000003
\$t2	10	0x00000004
\$t3	11	0x00000000

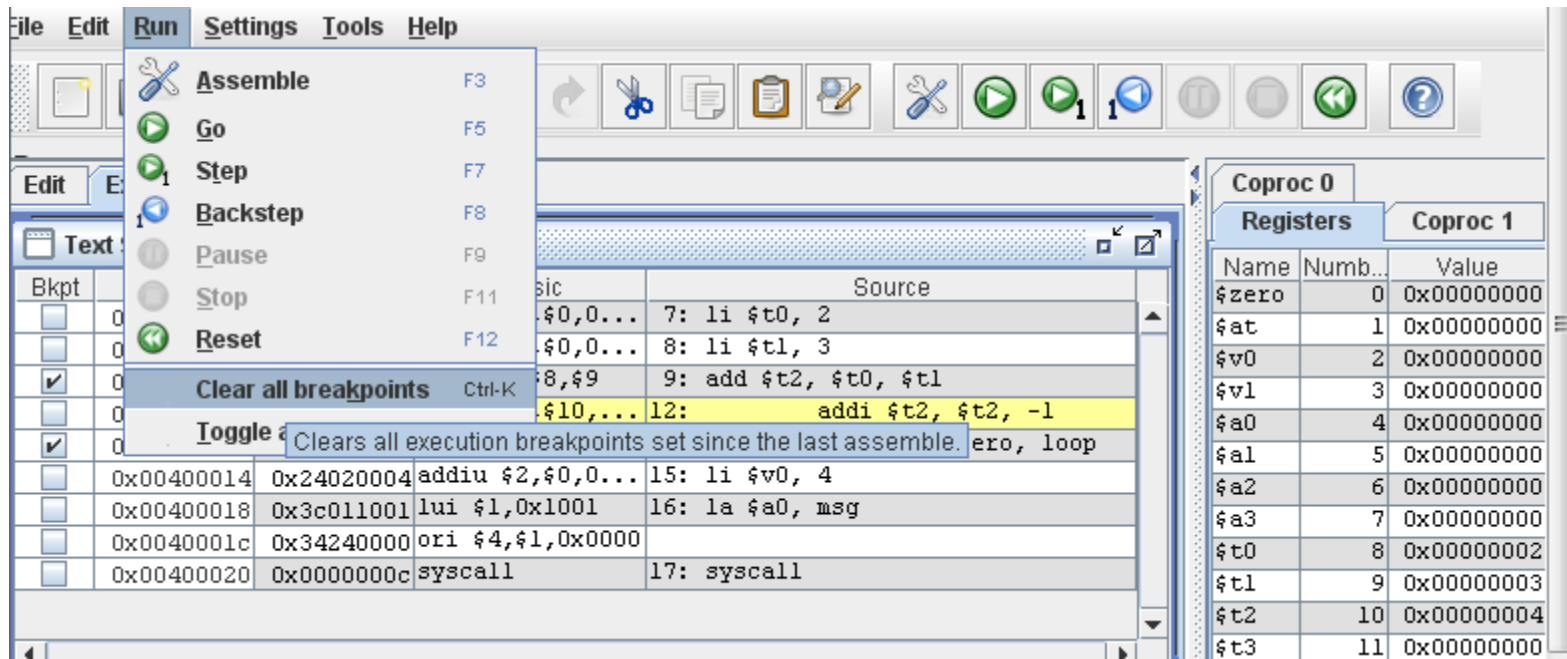
MARS Breakpoint

- All the breakpoints can be toggled between **Enabled** and **Disabled** status using the menu command *Run->Toggle all breakpoints*.



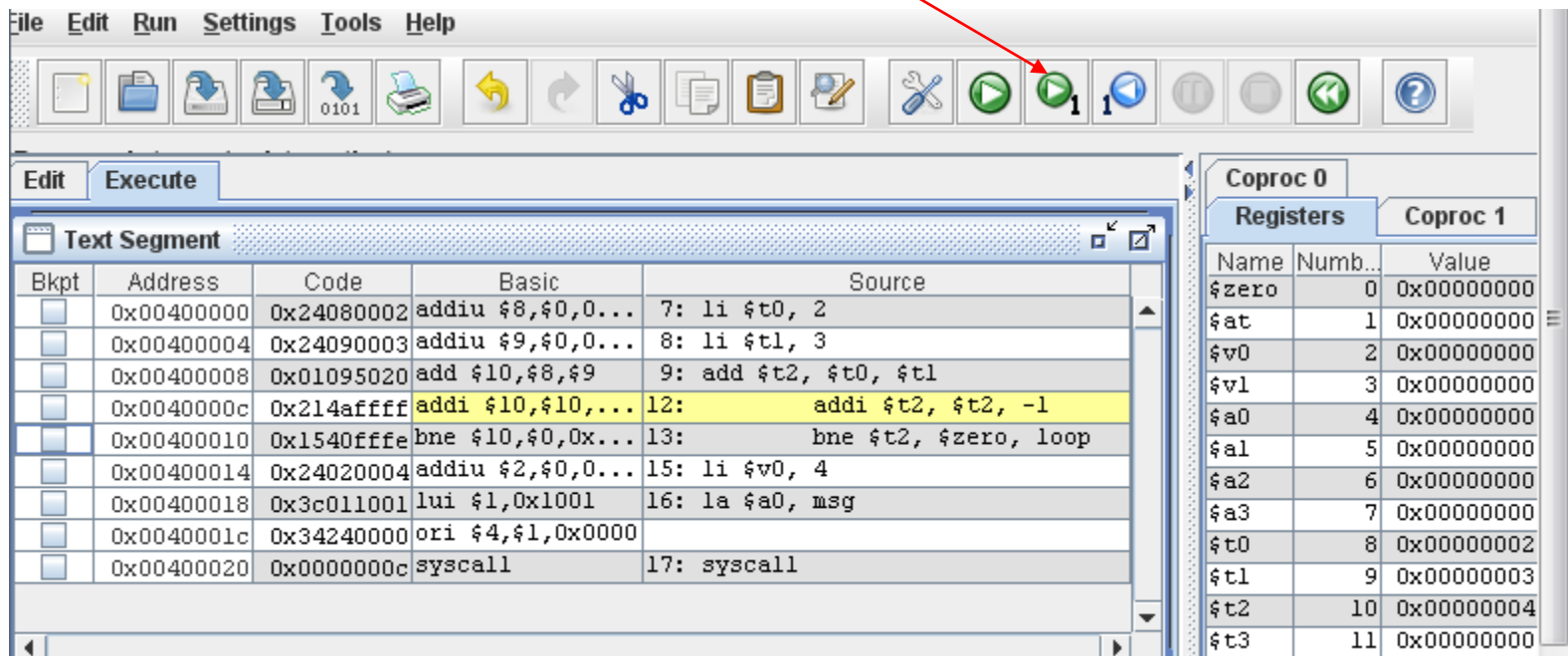
MARS Breakpoint

- All the breakpoints can be cleared out using the menu command *Run->Clear all breakpoints*.
- An individual breakpoint can also be cleared out by clicking on it to remove the tick on its box.



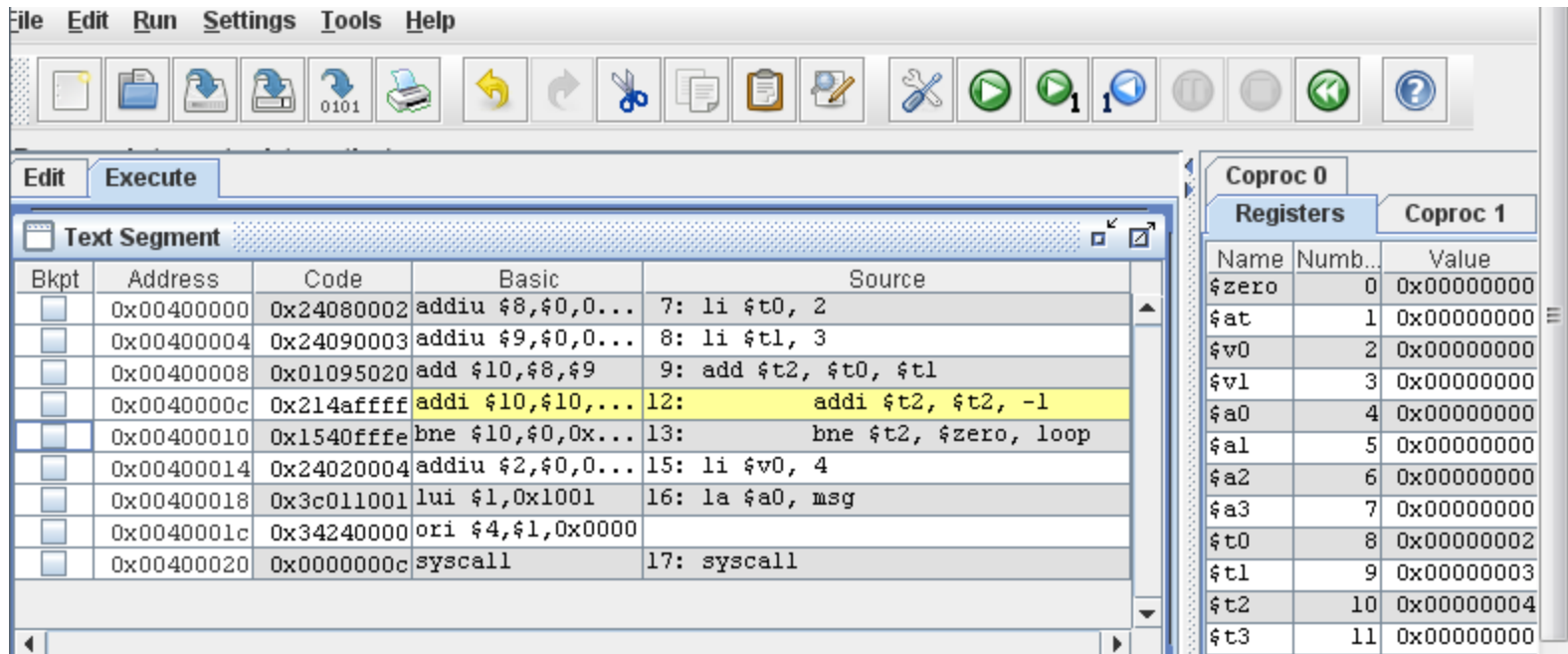
MARS Single Step

- Single Step – executes one instruction and then pauses the execution.
 - ❑ Click this **Single Step** button to do a Single Step.
 - ❑ It can be used at the very beginning to start executing the program (executing its first instruction) or at any time when the program execution is paused (e.g., by a breakpoint).



MARS Single Step

- Try executing each instruction in `debug1.s` using Single Step.
- See how the execution of the loop in `debug1.s` is traced in this way.
- See how the values of the registers and/or memory are changed to reflect the latest execution.



The screenshot shows the MARS interface with the following components:

- Menu Bar:** File, Edit, Run, Settings, Tools, Help
- Toolbar:** Contains icons for file operations (new, open, save, print), navigation (back, forward, search), and execution (single step, run, stop, reset).
- Text Segment Table:**

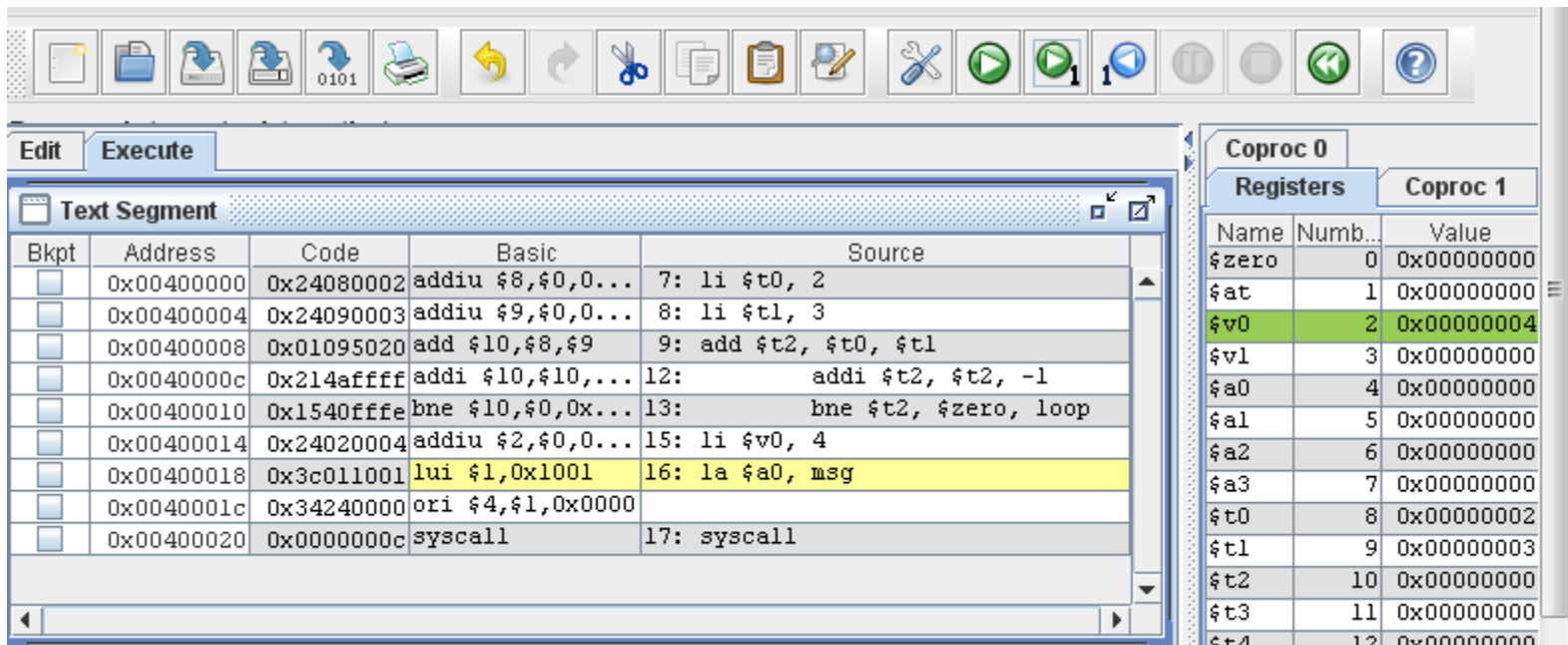
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x24080002	addiu \$8,\$0,0...	7: li \$t0, 2
<input type="checkbox"/>	0x00400004	0x24090003	addiu \$9,\$0,0...	8: li \$t1, 3
<input type="checkbox"/>	0x00400008	0x01095020	add \$10,\$8,\$9	9: add \$t2, \$t0, \$t1
<input type="checkbox"/>	0x0040000c	0x214affff	addi \$10,\$10,...	12: addi \$t2, \$t2, -1
<input type="checkbox"/>	0x00400010	0x1540fffe	bne \$10,\$0,0x...	13: bne \$t2, \$zero, loop
<input type="checkbox"/>	0x00400014	0x24020004	addiu \$2,\$0,0...	15: li \$v0, 4
<input type="checkbox"/>	0x00400018	0x3c011001	lui \$1,0x1001	16: la \$a0, msg
<input type="checkbox"/>	0x0040001c	0x34240000	ori \$4,\$1,0x0000	
<input type="checkbox"/>	0x00400020	0x0000000c	syscall	17: syscall

- Registers Panel (Coproc 0):**

Name	Numb...	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000002
\$t1	9	0x00000003
\$t2	10	0x00000004
\$t3	11	0x00000000

MARS Single Step

- The register whose value has just been changed by an instruction execution is highlighted in green at that moment.



The screenshot displays the MARS Single Step debugger interface. The main window shows a list of instructions in a table. The instruction `lui $1, 0x1001` at address `0x00400018` is highlighted in yellow. To the right, the 'Registers' panel shows the value of register `$v0` highlighted in green.

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x24080002	<code>addiu \$8,\$0,0...</code>	7: <code>li \$t0, 2</code>
<input type="checkbox"/>	0x00400004	0x24090003	<code>addiu \$9,\$0,0...</code>	8: <code>li \$t1, 3</code>
<input type="checkbox"/>	0x00400008	0x01095020	<code>add \$10,\$8,\$9</code>	9: <code>add \$t2, \$t0, \$t1</code>
<input type="checkbox"/>	0x0040000c	0x214affff	<code>addi \$10,\$10,...</code>	12: <code>addi \$t2, \$t2, -1</code>
<input type="checkbox"/>	0x00400010	0x1540fffe	<code>bne \$10,\$0,0x...</code>	13: <code>bne \$t2, \$zero, loop</code>
<input type="checkbox"/>	0x00400014	0x24020004	<code>addiu \$2,\$0,0...</code>	15: <code>li \$v0, 4</code>
<input type="checkbox"/>	0x00400018	0x3c011001	<code>lui \$1,0x1001</code>	16: <code>la \$a0, msg</code>
<input type="checkbox"/>	0x0040001c	0x34240000	<code>ori \$4,\$1,0x0000</code>	
<input type="checkbox"/>	0x00400020	0x0000000c	<code>syscall</code>	17: <code>syscall</code>

Coproc 0		
Registers		
Name	Numb...	Value
<code>\$zero</code>	0	0x00000000
<code>\$at</code>	1	0x00000000
<code>\$v0</code>	2	0x00000004
<code>\$v1</code>	3	0x00000000
<code>\$a0</code>	4	0x00000000
<code>\$a1</code>	5	0x00000000
<code>\$a2</code>	6	0x00000000
<code>\$a3</code>	7	0x00000000
<code>\$t0</code>	8	0x00000002
<code>\$t1</code>	9	0x00000003
<code>\$t2</code>	10	0x00000000
<code>\$t3</code>	11	0x00000000
<code>\$t4</code>	12	0x00000000

MARS Single Step

- See how the console window reflects the Input/Output in the latest execution of an I/O syscall.

The screenshot displays the MARS interface during a single-step execution of a program. The top window shows the assembly code with the instruction `syscall` at address `0x00400020`. Below this, the **Data Segment** window shows a memory dump with addresses and values. The **Registers** window on the right lists registers `$t0` through `$ra` with their current values. The **Mars Messages** window at the bottom shows the message `loop ended!`.

Assembly Code:

Address	Value	Comment
0x00400020	0x0000000c	syscall

Data Segment:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x100...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...
0x100...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...
0x100...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...
0x100...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...
0x100...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...
0x100...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...
0x100...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...
0x100...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...

Registers:

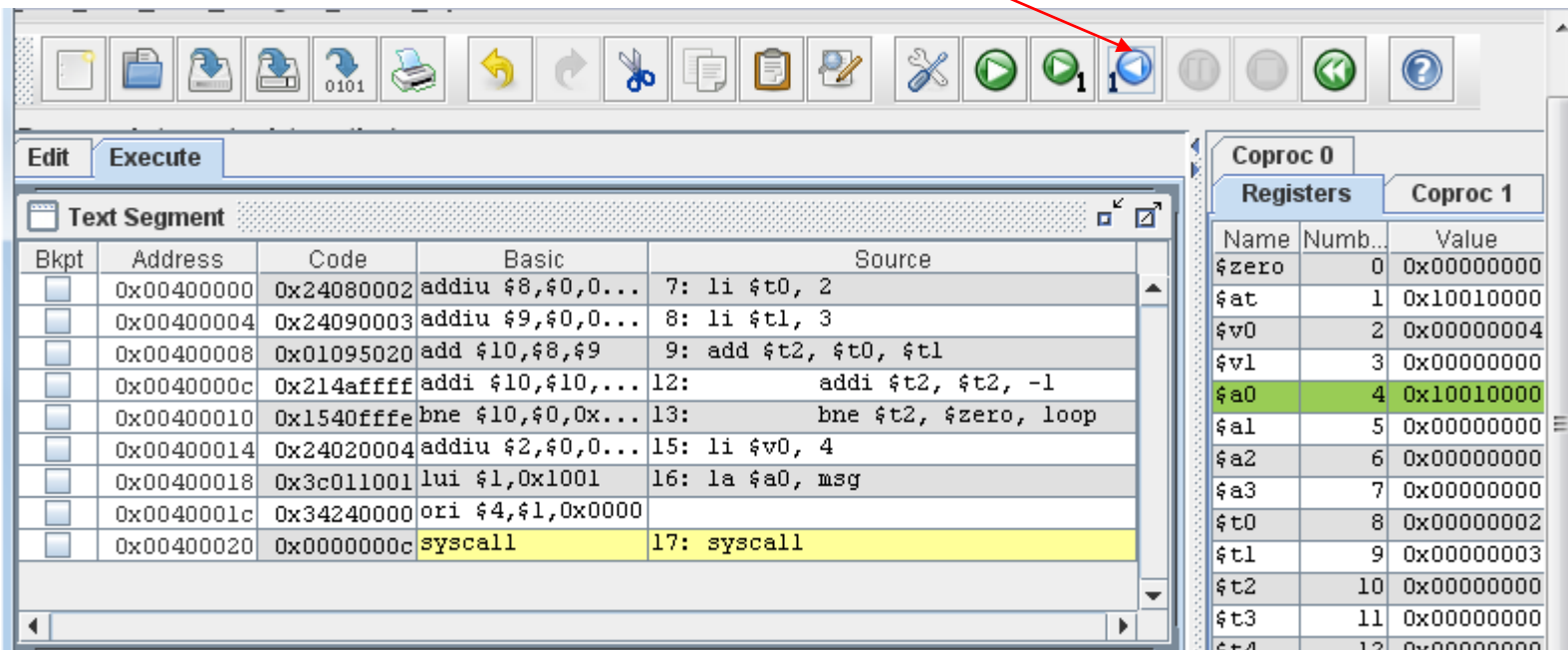
Register	Value
\$t0	8 0x00000002
\$t1	9 0x00000003
\$t2	10 0x00000000
\$t3	11 0x00000000
\$t4	12 0x00000000
\$t5	13 0x00000000
\$t6	14 0x00000000
\$t7	15 0x00000000
\$s0	16 0x00000000
\$s1	17 0x00000000
\$s2	18 0x00000000
\$s3	19 0x00000000
\$s4	20 0x00000000
\$s5	21 0x00000000
\$s6	22 0x00000000
\$s7	23 0x00000000
\$t8	24 0x00000000
\$t9	25 0x00000000
\$k0	26 0x00000000
\$k1	27 0x00000000
\$gp	28 0x10008000
\$sp	29 0x7ffffeffc
\$fp	30 0x00000000
\$ra	31 0x00000000

Mars Messages:

loop ended!

MARS Undo Step

- Undo Step – undo the last instruction (up to a maximum of 2000 instructions by default) and then pauses the execution.
 - ❑ Click this **Undo Step** button to do an Undo Step.
 - ❑ It can only be used when the program execution is paused.
 - ❑ It can still be used after the program execution terminated (but before it is reset). Try this after executing `debug1.s`.



MARS Undo Step

- Try reversing the execution of the loop in [debug1.s](#), too.
- See how the values of the registers and memory are changed back as the execution reverses.

The screenshot displays the MARS MIPS simulator interface. The top toolbar contains various icons for file operations, execution, and debugging. Below the toolbar, the 'Execute' tab is active, showing the 'Text Segment' panel. This panel contains a table of assembly instructions with columns for 'Bkpt', 'Address', 'Code', 'Basic', and 'Source'. The instruction at address 0x00400010, 'bne \$t0, \$0, 0x...', is highlighted in yellow. To the right of the 'Text Segment' panel, the 'Registers' panel is visible, showing the state of MIPS registers. The register '\$t2' is highlighted in green and contains the value 0x00000001.

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x24080002	addiu \$8,\$0,0...	7: li \$t0, 2
<input type="checkbox"/>	0x00400004	0x24090003	addiu \$9,\$0,0...	8: li \$t1, 3
<input type="checkbox"/>	0x00400008	0x01095020	add \$10,\$8,\$9	9: add \$t2, \$t0, \$t1
<input type="checkbox"/>	0x0040000c	0x214affff	addi \$10,\$10,...	12: addi \$t2, \$t2, -1
<input type="checkbox"/>	0x00400010	0x1540fffe	bne \$10,\$0,0x...	13: bne \$t2, \$zero, loop
<input type="checkbox"/>	0x00400014	0x24020004	addiu \$2,\$0,0...	15: li \$v0, 4
<input type="checkbox"/>	0x00400018	0x3c011001	lui \$1,0x1001	16: la \$a0, msg
<input type="checkbox"/>	0x0040001c	0x34240000	ori \$4,\$1,0x0000	
<input type="checkbox"/>	0x00400020	0x0000000c	syscall	17: syscall

Coproc 0		
Registers		
Name	Numb...	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000002
\$t1	9	0x00000003
\$t2	10	0x00000001
\$t3	11	0x00000000

MARS Undo Step

- Now execute the program forward again. You will get another output of "loop ended!" on the I/O console.

The screenshot displays the MARS MIPS simulator interface. The top panel shows the assembly code with the following instructions:

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x24080002	addiu \$8,\$0,0...	7: li \$t0, 2
<input type="checkbox"/>	0x00400004	0x24090003	addiu \$9,\$0,0...	8: li \$t1, 3
<input type="checkbox"/>	0x00400008	0x01095020	add \$10,\$8,\$9	9: add \$t2, \$t0, \$t1
<input type="checkbox"/>	0x0040000c	0x214affff	addi \$10,\$10,...	12: addi \$t2, \$t2, -1
<input type="checkbox"/>	0x00400010	0x1540fffe	bne \$10,\$0,0x...	13: bne \$t2, \$zero, loop
<input type="checkbox"/>	0x00400014	0x24020004	addiu \$2,\$0,0...	15: li \$v0, 4
<input type="checkbox"/>	0x00400018	0x3c011001	lui \$1,0x1001	16: la \$a0, msg
<input type="checkbox"/>	0x0040001c	0x34240000	ori \$4,\$1,0x0000	
<input type="checkbox"/>	0x00400020	0x0000000c	syscall	17: syscall

The middle panel shows the Data Segment with the following values:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x100...	0x706...	0x646...	0x002...	0x000...	0x000...	0x000...	0x000...	0x000...
0x100...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...
0x100...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...
0x100...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...
0x100...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...
0x100...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...
0x100...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...
0x100...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...

The right panel shows the register values:

Register	Value
\$zero	0 0x00000000
\$at	1 0x10010000
\$v0	2 0x00000004
\$v1	3 0x00000000
\$a0	4 0x10010000
\$a1	5 0x00000000
\$a2	6 0x00000000
\$a3	7 0x00000000
\$t0	8 0x00000002
\$t1	9 0x00000003
\$t2	10 0x00000000
\$t3	11 0x00000000
\$t4	12 0x00000000
\$t5	13 0x00000000
\$t6	14 0x00000000
\$t7	15 0x00000000
\$s0	16 0x00000000
\$s1	17 0x00000000
\$s2	18 0x00000000
\$s3	19 0x00000000
\$s4	20 0x00000000
\$s5	21 0x00000000
\$s6	22 0x00000000
\$s7	23 0x00000000
\$t8	24 0x00000000
\$t9	25 0x00000000
\$k0	26 0x00000000
\$k1	27 0x00000000
\$gp	28 0x10008000
\$sp	29 0x7fffffc
\$fp	30 0x00000000
\$ra	31 0x00000000

The bottom panel shows the Mars Messages window with the text "loop ended!loop ended!" and the Run I/O button.

MARS Undo Step

- Undo Step can only undo the change in a register or memory but not in the syscall service such as Console I/O that has already been performed.

Modifying registers or memory

- When you figure out a possible solution to fix a buggy program, you can modify the program code to try it out.
- You can also just modify the values of the registers or memory (according to the solution) during the (buggy) program execution.
- This lets you get a sense of whether the solution should work before you modify any codes.
- To modify a register or memory,
 - ☐ double-click on it on the Registers or Data Segment window.
 - ☐ Type the new value in hexadecimal or decimal format.
 - ☐ Finally, press the enter key to apply the new value.
- The modification can only be done before the program execution starts or when it is paused.
- The new value will be applied to all the subsequent executions.

Modifying registers

- Try executing the program debug1.s.
- Then pause it at the instruction in Line 9 and modify the value of the register t0 to 0x0000001a (as shown by the image below).

Text Segment					Registers			Coproc 1
Bkpt	Address	Code	Basic	Source	Name	Numb...	Value	
<input type="checkbox"/>	0x00400000	0x24080002	addiu \$8,\$0,0...	7: li \$t0, 2	\$zero	0	0x00000000	
<input type="checkbox"/>	0x00400004	0x24090003	addiu \$9,\$0,0...	8: li \$t1, 3	\$at	1	0x00000000	
<input type="checkbox"/>	0x00400008	0x01095020	add \$10,\$8,\$9	9: add \$t2, \$t0, \$t1	\$v0	2	0x00000000	
<input type="checkbox"/>	0x0040000c	0x214affff	addi \$10,\$10,...	12: addi \$t2, \$t2, -1	\$v1	3	0x00000000	
<input type="checkbox"/>	0x00400010	0x1540fffe	bne \$10,\$0,0x...	13: bne \$t2, \$zero, loop	\$a0	4	0x00000000	
<input type="checkbox"/>	0x00400014	0x24020004	addiu \$2,\$0,0...	15: li \$v0, 4	\$a1	5	0x00000000	
<input type="checkbox"/>	0x00400018	0x3c011001	lui \$1,0x1001	16: la \$a0, msg	\$a2	6	0x00000000	
<input type="checkbox"/>	0x0040001c	0x34240000	ori \$4,\$1,0x0000		\$a3	7	0x00000000	
<input type="checkbox"/>	0x00400020	0x0000000c	syscall	17: syscall	\$t0	8	0x0000001a	
					\$t1	9	0x00000003	
					\$t2	10	0x00000000	
					\$t3	11	0x00000000	

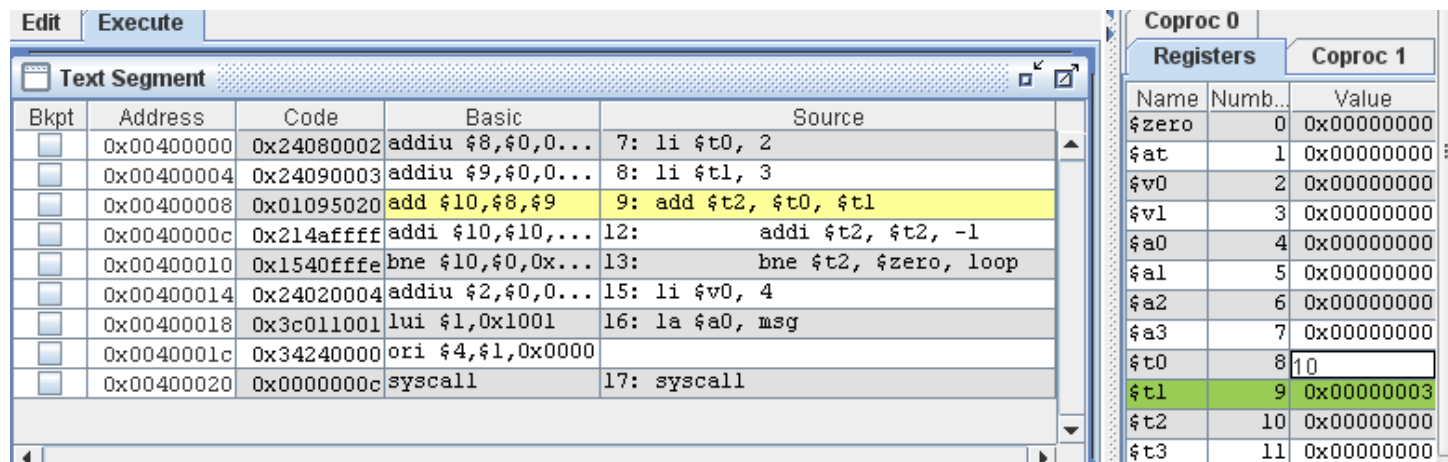
Modifying registers

- Single Step the add instruction in Line 9.
- See how the addition used the new value of the register t0 (look at the sum in the register t2).

Edit Execute					Coprocc 0		
Text Segment					Registers		Coprocc 1
Bkpt	Address	Code	Basic	Source	Name	Numb...	Value
<input type="checkbox"/>	0x00400000	0x24080002	addiu \$8,\$0,0...	7: li \$t0, 2	\$zero	0	0x00000000
<input type="checkbox"/>	0x00400004	0x24090003	addiu \$9,\$0,0...	8: li \$t1, 3	\$at	1	0x00000000
<input type="checkbox"/>	0x00400008	0x01095020	add \$10,\$8,\$9	9: add \$t2, \$t0, \$t1	\$v0	2	0x00000000
<input type="checkbox"/>	0x0040000c	0x214affff	addi \$10,\$10,...	12: addi \$t2, \$t2, -1	\$v1	3	0x00000000
<input type="checkbox"/>	0x00400010	0x1540fffe	bne \$10,\$0,0x...	13: bne \$t2, \$zero, loop	\$a0	4	0x00000000
<input type="checkbox"/>	0x00400014	0x24020004	addiu \$2,\$0,0...	15: li \$v0, 4	\$a1	5	0x00000000
<input type="checkbox"/>	0x00400018	0x3c011001	lui \$1,0x1001	16: la \$a0, msg	\$a2	6	0x00000000
<input type="checkbox"/>	0x0040001c	0x34240000	ori \$4,\$1,0x0000		\$a3	7	0x00000000
<input type="checkbox"/>	0x00400020	0x0000000c	syscall	17: syscall	\$t0	8	0x0000001a
					\$t1	9	0x00000003
					\$t2	10	0x0000001d
					\$t3	11	0x00000000

Modifying registers

- New decimal values entered will be converted to the display format of the Registers window (hexadecimal by default).

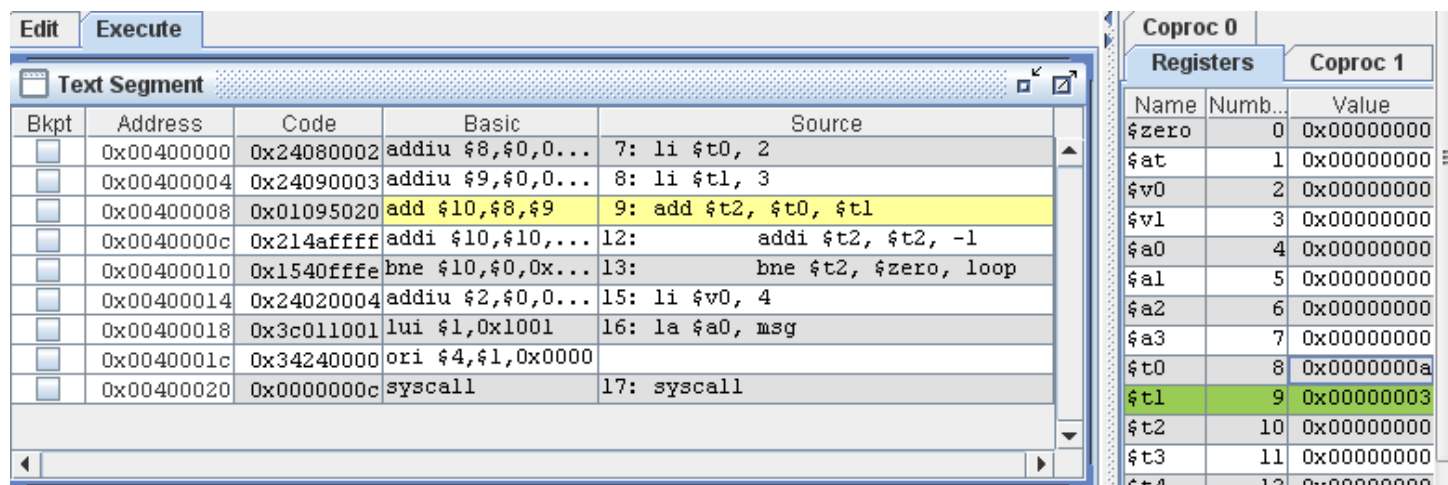


The screenshot shows the debugger interface with the 'Text Segment' window and the 'Registers' window. The 'Text Segment' window displays the following assembly code:

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x24080002	addiu \$8,\$0,0...	7: li \$t0, 2
<input type="checkbox"/>	0x00400004	0x24090003	addiu \$9,\$0,0...	8: li \$t1, 3
<input type="checkbox"/>	0x00400008	0x01095020	add \$t0,\$8,\$9	9: add \$t2, \$t0, \$t1
<input type="checkbox"/>	0x0040000c	0x214affff	addi \$t0,\$t0,...	12: addi \$t2, \$t2, -1
<input type="checkbox"/>	0x00400010	0x1540fffe	bne \$t0,\$0,0x...	13: bne \$t2, \$zero, loop
<input type="checkbox"/>	0x00400014	0x24020004	addiu \$2,\$0,0...	15: li \$v0, 4
<input type="checkbox"/>	0x00400018	0x3c011001	lui \$l,0x1001	16: la \$a0, msg
<input type="checkbox"/>	0x0040001c	0x34240000	ori \$4,\$l,0x0000	
<input type="checkbox"/>	0x00400020	0x0000000c	syscall	17: syscall

The 'Registers' window shows the current values of registers \$zero through \$t3:

Name	Numb...	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0
\$t1	9	0x00000003
\$t2	10	0x00000000
\$t3	11	0x00000000



The screenshot shows the debugger interface with the 'Text Segment' window and the 'Registers' window. The 'Text Segment' window displays the following assembly code:

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x24080002	addiu \$8,\$0,0...	7: li \$t0, 2
<input type="checkbox"/>	0x00400004	0x24090003	addiu \$9,\$0,0...	8: li \$t1, 3
<input type="checkbox"/>	0x00400008	0x01095020	add \$t0,\$8,\$9	9: add \$t2, \$t0, \$t1
<input type="checkbox"/>	0x0040000c	0x214affff	addi \$t0,\$t0,...	12: addi \$t2, \$t2, -1
<input type="checkbox"/>	0x00400010	0x1540fffe	bne \$t0,\$0,0x...	13: bne \$t2, \$zero, loop
<input type="checkbox"/>	0x00400014	0x24020004	addiu \$2,\$0,0...	15: li \$v0, 4
<input type="checkbox"/>	0x00400018	0x3c011001	lui \$l,0x1001	16: la \$a0, msg
<input type="checkbox"/>	0x0040001c	0x34240000	ori \$4,\$l,0x0000	
<input type="checkbox"/>	0x00400020	0x0000000c	syscall	17: syscall

The 'Registers' window shows the current values of registers \$zero through \$t4:

Name	Numb...	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x0000000a
\$t1	9	0x00000003
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000

Modifying registers

- The hexadecimal format on MARS uses 2's complement.

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x24080002	addiu \$8,\$0,0...	7: li \$t0, 2
<input type="checkbox"/>	0x00400004	0x24090003	addiu \$9,\$0,0...	8: li \$t1, 3
<input type="checkbox"/>	0x00400008	0x01095020	add \$t0,\$8,\$9	9: add \$t2, \$t0, \$t1
<input type="checkbox"/>	0x0040000c	0x214affff	addi \$t0,\$t0,...	12: addi \$t2, \$t2, -1
<input type="checkbox"/>	0x00400010	0x1540fffe	bne \$t0,\$0,0x...	13: bne \$t2, \$zero, loop
<input type="checkbox"/>	0x00400014	0x24020004	addiu \$2,\$0,0...	15: li \$v0, 4
<input type="checkbox"/>	0x00400018	0x3c011001	lui \$l,0x1001	16: la \$a0, msg
<input type="checkbox"/>	0x0040001c	0x34240000	ori \$4,\$l,0x0000	
<input type="checkbox"/>	0x00400020	0x0000000c	syscall	17: syscall

Coproc 0 Registers

Name	Numb...	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000003
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x24080002	addiu \$8,\$0,0...	7: li \$t0, 2
<input type="checkbox"/>	0x00400004	0x24090003	addiu \$9,\$0,0...	8: li \$t1, 3
<input type="checkbox"/>	0x00400008	0x01095020	add \$t0,\$8,\$9	9: add \$t2, \$t0, \$t1
<input type="checkbox"/>	0x0040000c	0x214affff	addi \$t0,\$t0,...	12: addi \$t2, \$t2, -1
<input type="checkbox"/>	0x00400010	0x1540fffe	bne \$t0,\$0,0x...	13: bne \$t2, \$zero, loop
<input type="checkbox"/>	0x00400014	0x24020004	addiu \$2,\$0,0...	15: li \$v0, 4
<input type="checkbox"/>	0x00400018	0x3c011001	lui \$l,0x1001	16: la \$a0, msg
<input type="checkbox"/>	0x0040001c	0x34240000	ori \$4,\$l,0x0000	
<input type="checkbox"/>	0x00400020	0x0000000c	syscall	17: syscall

Coproc 0 Registers

Name	Numb...	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0xffffffffe
\$t1	9	0x00000003
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000

Modifying memory

- For example, modify the string msg in `debug1.s`.

Data Segment								
Address	Value (+0)	Value (+4)	Value (+...	Value (...)	Value (+...	Value (...)	Value (...)	Value (...)
0x10010000	0x706f6f6f	0x646e6520	0x00...	0x0...	0x00...	0x00...	0x00...	0x00...
0x10010020	0x00000000	0x00000000	0x00...	0x0...	0x00...	0x00...	0x00...	0x00...
0x10010040	0x00000000	0x00000000	0x00...	0x0...	0x00...	0x00...	0x00...	0x00...
0x10010060	0x00000000	0x00000000	0x00...	0x0...	0x00...	0x00...	0x00...	0x00...
0x10010080	0x00000000	0x00000000	0x00...	0x0...	0x00...	0x00...	0x00...	0x00...
0x100100a0	0x00000000	0x00000000	0x00...	0x0...	0x00...	0x00...	0x00...	0x00...

Data Segment								
Address	Value (+0)	Value (+4)	Value (+...	Value (...)	Value (+...	Value (...)	Value (...)	Value (...)
0x10010000	0x706f6f6f	0x646e6520	0x00...	0x0...	0x00...	0x00...	0x00...	0x00...
0x10010020	0x00000000	0x00000000	0x00...	0x0...	0x00...	0x00...	0x00...	0x00...
0x10010040	0x00000000	0x00000000	0x00...	0x0...	0x00...	0x00...	0x00...	0x00...
0x10010060	0x00000000	0x00000000	0x00...	0x0...	0x00...	0x00...	0x00...	0x00...
0x10010080	0x00000000	0x00000000	0x00...	0x0...	0x00...	0x00...	0x00...	0x00...
0x100100a0	0x00000000	0x00000000	0x00...	0x0...	0x00...	0x00...	0x00...	0x00...
0x100100c0	0x00000000	0x00000000	0x00...	0x0...	0x00...	0x00...	0x00...	0x00...
0x100100e0	0x00000000	0x00000000	0x00...	0x0...	0x00...	0x00...	0x00...	0x00...

0x10010000 (.data)

☒ Hexadecimal Addresses ☒ Hexadecimal Values

Mars Messages

Run I/O

oops ended!
-- program is finished running (dropped off bottom) --

Clear

Example program

- Try to debug the example program `debug2.s`.
 - To pause the program execution (e.g., during its infinite loop), click this **Pause** button.

The screenshot shows the Mars MIPS simulator interface. The top toolbar contains various icons for file operations, execution, and debugging. A red arrow points to the **Pause** button, which is represented by a square button with a vertical bar in the center.

The main window is divided into several sections:

- Text Segment:** A table showing assembly code with columns for Bkpt, Address, Code, Basic, and Source.
- Data Segment:** A table showing memory addresses and their corresponding values.
- Mars Messages:** A section for displaying messages, currently showing "2 + 3 + 4 =".
- Registers:** A panel on the right showing the state of Coprocessor 0 registers.

The assembly code in the Text Segment is as follows:

Bkpt	Address	Code	Basic	Source
	0x00400000	0x00000000	syscall	10: syscall
	0x00400010	0x24110000	addiu \$t1,\$0,...	12: li \$s1, 0
	0x00400014	0x3c011001	lui \$t1,0x1001	13: la \$s0, list
	0x00400018	0x34300010	ori \$t1,\$t1,0x...	
	0x0040001c	0x24080003	addiu \$t1,\$0,0...	14: li \$t0, 3
	0x00400020	0x11000004	beq \$t1,\$0,0x0004	16: beq \$t0, \$zero, loop...
	0x00400024	0x8e090000	lw \$t1,0x0000(...	17: lw \$t1, 0(\$s0)
	0x00400028	0x02298820	add \$t1,\$t1,\$t1	18: add \$s1, \$s1, \$t1
	0x0040002c	0x2108ffff	addi \$t1,\$t1,0x...	19: addi \$t0, \$t0, -2
	0x00400030	0x08100008	j 0x00400020	20: j loop

The Data Segment table is as follows:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+1...	Value (+18)	Value (+1c)
0x100...	0x202...	0x202...	0x203...	0x000...	0x000...	0x000...	0x000...	0x000...

The Registers panel shows the state of Coprocessor 0 registers:

Name	Num...	Value
\$zero	0	0x00000000
\$at	1	0x10010000
\$v0	2	0x00000004
\$v1	3	0x00000000
\$a0	4	0x10010000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0xffffd68eb
\$t1	9	0x00000002
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x10010010
\$s1	17	0x00029718
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000

Example program

- After figuring out a possible solution to the bug, try it by modifying only the registers (not the program code) to get a sense of whether it should work (computing the correct sum).
- For example, see how the register modifications yield the correct result (in the register s1) of the addition done in each of the first three iterations of the loop in debug2.s.
- After a correct solution is found, fix the program code then.

The screenshot displays the Mars MIPS simulator interface. The assembly code window shows the following instructions:

Address	Hex	Assembly	Comment
0x00400010	0x24110000	addiu \$t7,\$0,...	12: li \$s1, 0
0x00400014	0x3c011001	lui \$l,0x1001	13: la \$s0, list
0x00400018	0x34300010	ori \$l6,\$l,0x...	
0x0040001c	0x24080003	addiu \$8,\$0,0...	14: li \$t0, 3
0x00400020	0x11000004	beq \$8,\$0,0x0004	16: beq \$t0, \$zero, loop...
0x00400024	0x8e090000	lw \$9,0x0000(...	17: lw \$t1, 0(\$s0)
0x00400028	0x02298820	add \$l7,\$l7,\$9	18: add \$s1, \$s1, \$t1
0x0040002c	0x2108fffe	addi \$8,\$8,0x...	19: addi \$t0, \$t0, -2
0x00400030	0x08100008	j 0x00400020	20: j loop

The Data Segment window shows the following values:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+1...	Value (+18)	Value (+1c)
0x100...	0x202...	0x202...	0x203...	0x000...	0x000...	0x000...	0x000...	0x000...

The Registers window shows the following values:

Register	Value
\$v0	2 0x0000000a
\$v1	3 0x00000000
\$a0	4 0x00000009
\$a1	5 0x00000000
\$a2	6 0x00000000
\$a3	7 0x00000000
\$t0	8 0x00000000
\$t1	9 0x00000004
\$t2	10 0x00000000
\$t3	11 0x00000000
\$t4	12 0x00000000
\$t5	13 0x00000000
\$t6	14 0x00000000
\$t7	15 0x00000000
\$s0	16 0x10010018
\$s1	17 0x00000009
\$s2	18 0x00000000
\$s3	19 0x00000000
\$s4	20 0x00000000
\$s5	21 0x00000000

The Mars Messages window shows the following message:

2 + 3 + 4 = 9

The Run button is highlighted, and the program is finished running.

Another example program

- Try to debug the program `debug3.s`.
 - ❑ You may debug it by modifying the registers or program code first (whichever way you feel efficient with).
 - ❑ The program execution may run for a while, looking like in an infinite loop initially, but will eventually terminates with an Exception error at a particular instruction code.
 - ❑ See the message about the Exception on MARS' Messages Window.
 - ❑ The Exception is about an invalid memory access by the instruction. What is the cause?

Exercise

- Try to debug the program `debug4.s`.
 - ❑ The program prompts the user to input a number to search in a list of pre-specified numbers and outputs whether it is found in the list.
 - ❑ However, if the input number is found in the list, the program execution terminates with an Exception error about an invalid Program Counter value.
 - ❑ See the message about the Exception on MARS' Messages Window.
 - ❑ Figure out the cause of the Exception and debug the program.

Exercise (optional)

- Try to debug the program `debug5.s` at your own time.
 - ☐ The program prompts the user to input a number and calculate the factorial of that number. However, the output is not correct.
 - ☐ Figure out the cause of the problem and debug the program.

Conclusion

- You have learnt:

- ☐ how to debug a MIPS program using the debugging features in MARS.