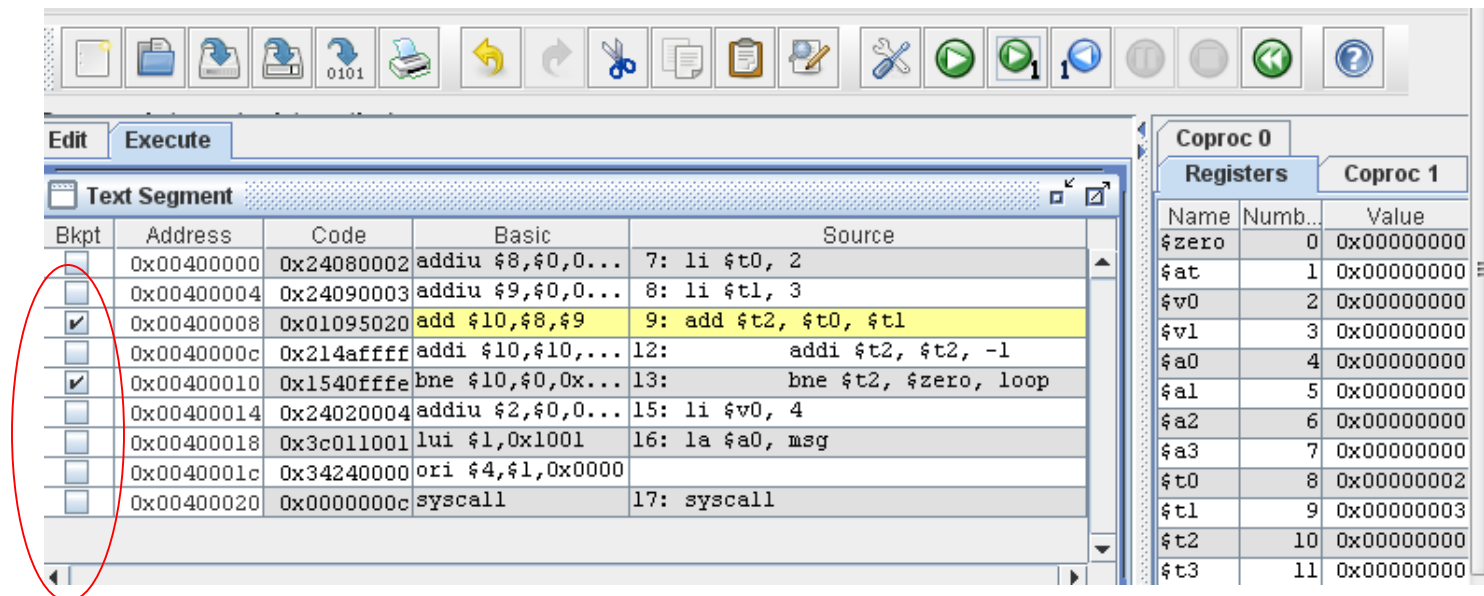


COMP2611: Computer Organization

Debugging on Mars

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

- ❑ 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.



The screenshot shows the Mars debugger interface. The main window displays a list of instructions in a table format. The columns are Bkpt, Address, Code, Basic, and Source. The instruction at address 0x00400008 is highlighted in yellow, and its Bkpt checkbox is checked. A red circle is drawn around the Bkpt column header and the checked checkbox. To the right of the instruction list, there is a panel for Coproc 0 Registers, showing a table of registers and their 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 \$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		Coproc 1
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

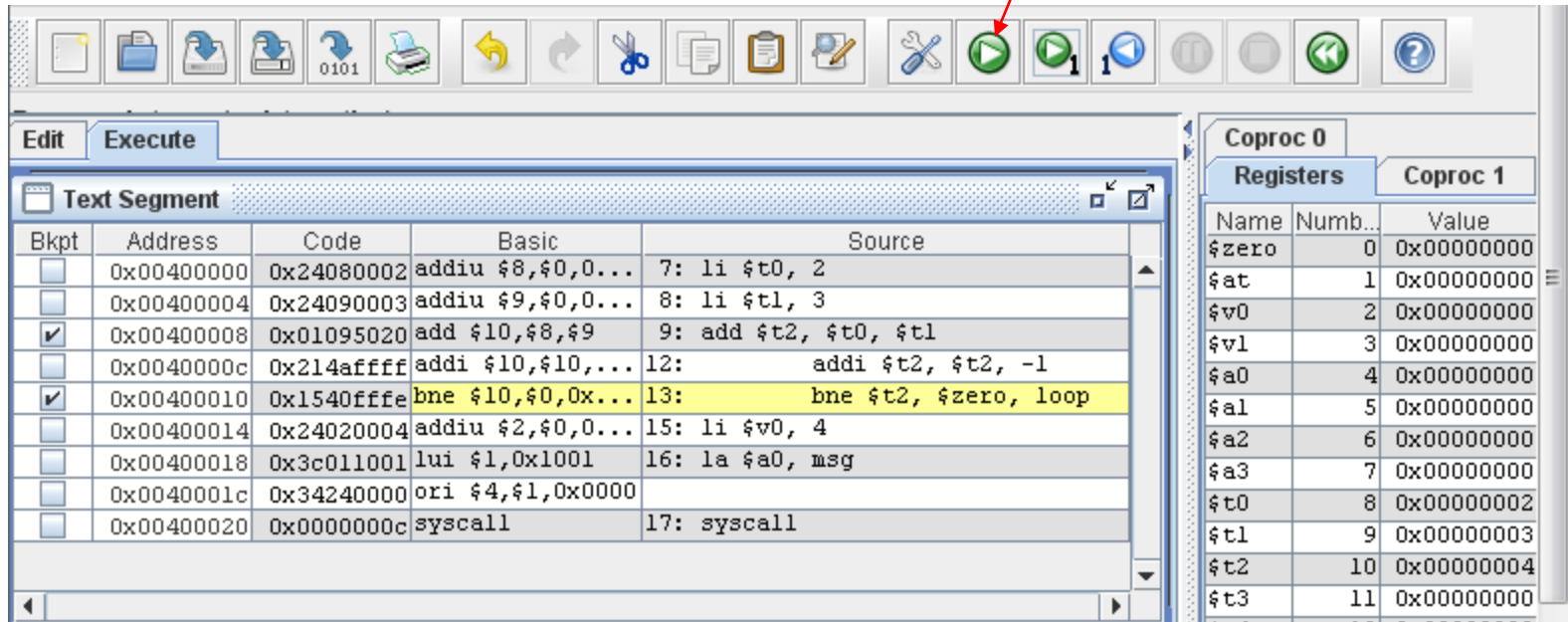
- Load the example program [debug1.s](#), and try enabling the breakpoints shown below.
- Then start the execution to see how it pauses at the first breakpoint.
- The values of the registers and memory reflect the execution up to (but not including) the instruction at this breakpoint.

The screenshot displays the Mars debugger interface. The main window shows a list of instructions in the 'Text Segment' with two breakpoints enabled. The instruction at address 0x00400008 is highlighted in yellow. To the right, the 'Registers' window shows the state of Coprocessor 0 registers.

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

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

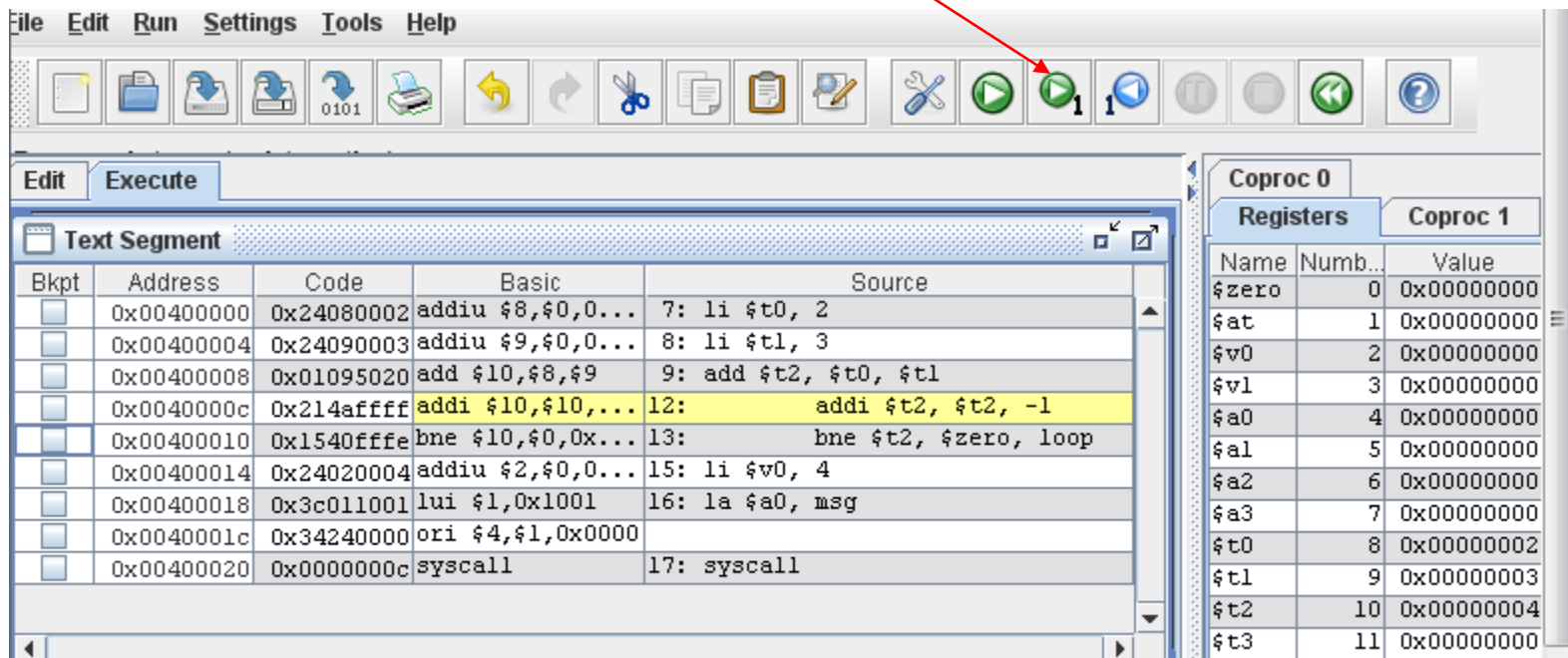


The screenshot shows the Mars debugger interface. At the top is a toolbar with various icons, including a green play button with a red arrow pointing to it. Below the toolbar is the 'Execute' tab, which contains a 'Text Segment' window. This window displays a list of instructions with their addresses, code, basic blocks, and source code. Two breakpoints are set, indicated by checked boxes in the 'Bkpt' column. The instruction at address 0x00400010 is highlighted in yellow. To the right of the 'Text Segment' window is the 'Registers' window, which shows the state of the processor registers for Coprocessor 0. The registers are listed with their names, numbers, and values.

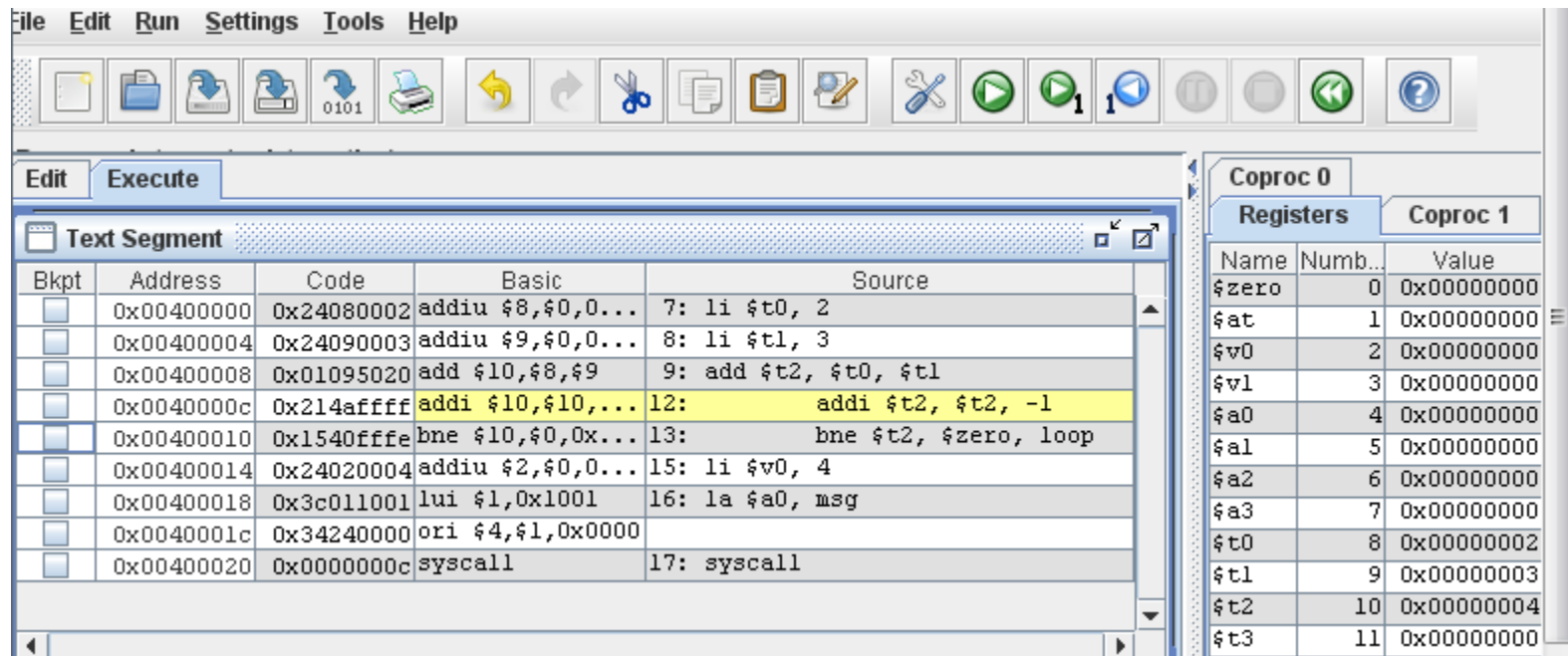
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x24080002	addiu \$8,\$0,0...	7: li \$t0, 2
<input type="checkbox"/>	0x00400004	0x24090003	addiu \$9,\$0,0...	8: li \$t1, 3
<input checked="" type="checkbox"/>	0x00400008	0x01095020	add \$10,\$8,\$9	9: add \$t2, \$t0, \$t1
<input type="checkbox"/>	0x0040000c	0x214affff	addi \$10,\$10,...	12: addi \$t2, \$t2, -1
<input checked="" type="checkbox"/>	0x00400010	0x1540fffe	bne \$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	0x00000004
\$t3	11	0x00000000

- ❑ 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).



- 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.

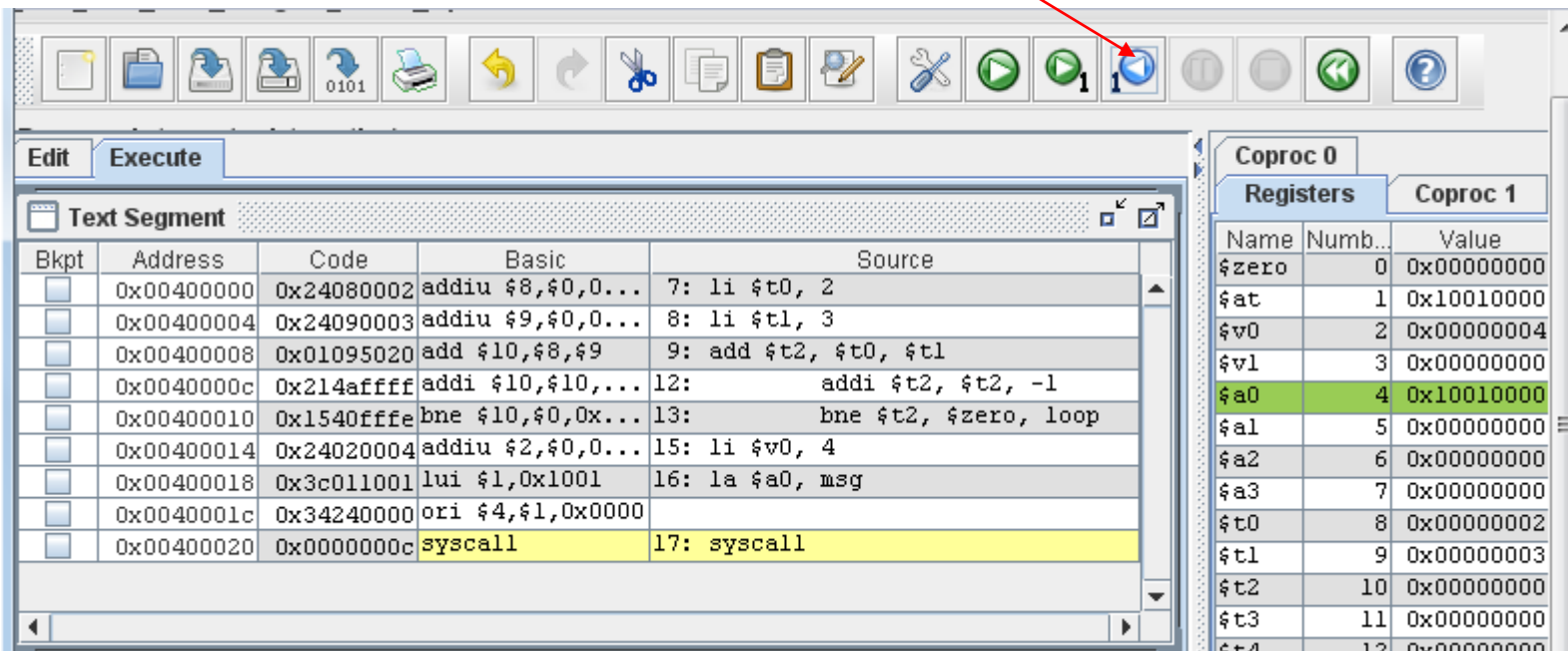


The screenshot shows the Mars debugger interface. The main window displays a list of instructions with their addresses, codes, basic forms, and source code. The instruction at address 0x0040000c is highlighted in yellow. To the right, a 'Registers' window shows the current values of registers \$zero through \$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 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	0x00000004
\$t3	11	0x00000000

- ❑ 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.



- 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 debugger interface. At the top is a toolbar with various icons for file operations, execution, and navigation. Below the toolbar are two main panels. The left panel, titled 'Text Segment', shows a list of assembly instructions with columns for 'Bkpt', 'Address', 'Code', 'Basic', and 'Source'. The instruction at address 0x00400010 is highlighted in yellow. The right panel, titled 'Coproc 0 Registers', shows a table of register values. The register \$t2 is highlighted in green.

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

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

- ❑ 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.

- ❑ 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).

The image shows a debugger window with two panes. The left pane, titled 'Text Segment', displays assembly code with columns for 'Bkpt', 'Address', 'Code', 'Basic', and 'Source'. The right pane, titled 'Registers', shows the state of Coprocessor 1 registers.

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

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	0x0000001a
\$t1	9	0x00000003
\$t2	10	0x00000000
\$t3	11	0x00000000

- ❑ 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).

The screenshot displays the Mars debugger interface. The main window shows a list of instructions in a table format. The instruction at line 12, `addi $t2, $t2, -1`, is highlighted in yellow. To the right, the 'Registers' window shows the state of Coprocessor 0 registers. Register `$t2` is highlighted in green, showing its value as `0x0000001d`.

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	0x00000000
<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	0x0000001a
<code>\$t1</code>	9	0x00000003
<code>\$t2</code>	10	0x0000001d
<code>\$t3</code>	11	0x00000000

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

The screenshot shows a debugger window with two panes. The left pane, titled 'Text Segment', displays a list of instructions with columns for Bkpt, Address, Code, Basic, and Source. The instruction at address 0x00400008 is highlighted in yellow: `add $t0,$8,$9` with source `9: add $t2, $t0, $t1`. The right pane, titled 'Registers', shows a table of registers for Coproc 0 and Coproc 1. Register `$t1` (number 9) is highlighted in green and has a value of 10.

Text Segment				
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 \$t0,\$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>

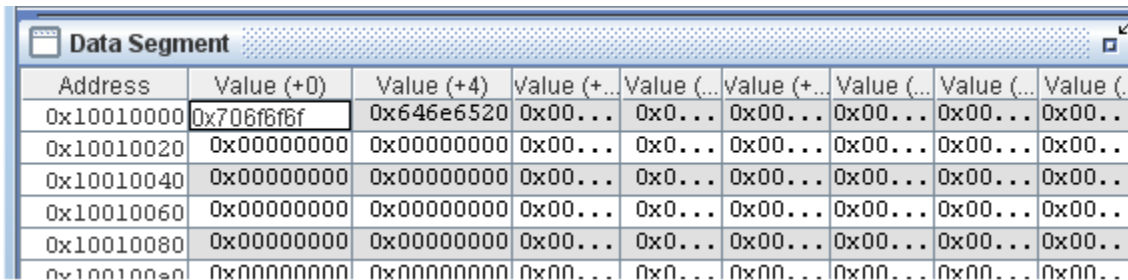
Registers		
Name	Numb...	Value
<code>\$zero</code>	0	0x00000000
<code>\$at</code>	1	0x00000000
<code>\$v0</code>	2	0x00000000
<code>\$v1</code>	3	0x00000000
<code>\$a0</code>	4	0x00000000
<code>\$a1</code>	5	0x00000000
<code>\$a2</code>	6	0x00000000
<code>\$a3</code>	7	0x00000000
<code>\$t0</code>	8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

This screenshot is identical to the one above, but the value of register `$t1` has been updated to 0x00000003, which is displayed in green text. The rest of the interface, including the text segment and other registers, remains the same.

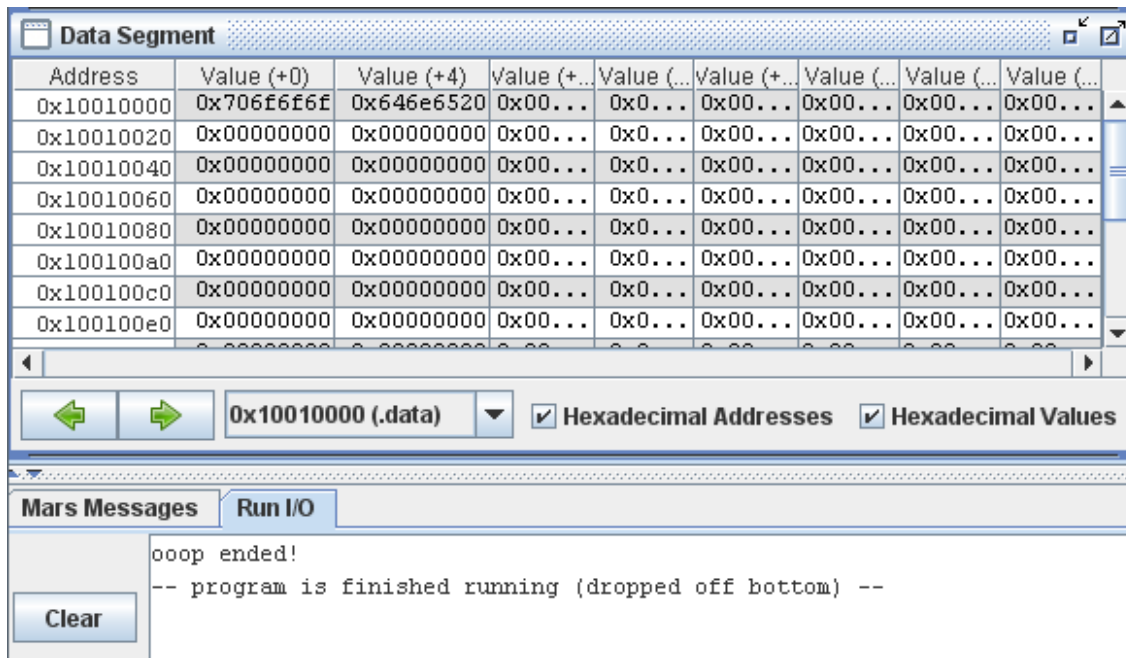
Text Segment				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x24080002	<code>addiu \$8,\$0,0...</code>	7: <code>li \$t0, 2</code>
<input type="checkbox"/>	0x00400004	0x24090003	<code>addiu \$9,\$0,0...</code>	8: <code>li \$t1, 3</code>
<input type="checkbox"/>	0x00400008	0x01095020	<code>add \$t0,\$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>

Registers		
Name	Numb...	Value
<code>\$zero</code>	0	0x00000000
<code>\$at</code>	1	0x00000000
<code>\$v0</code>	2	0x00000000
<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	0x0000000a
<code>\$t1</code>	9	0x00000003
<code>\$t2</code>	10	0x00000000
<code>\$t3</code>	11	0x00000000
<code>\$t4</code>	12	0x00000000

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



Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
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...



Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
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

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

Clear

- 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, with a red arrow pointing to the 'Pause' button (a grey circle with a vertical bar). The main window is divided into several panes:

- Text Segment:** A table showing assembly code with columns for Bkpt, Address, Code, Basic, and Source.
- Data Segment:** A table showing memory values with columns for Address and Value (+0), (+4), (+8), (+c), (+10), (+1..), (+18), (+1c).
- Mars Messages:** A pane showing the output of the program: `2 + 3 + 4 =`.
- Registers:** A pane showing the state of Coprocessor 0 registers, including Name, Num..., and Value.

Bkpt	Address	Code	Basic	Source
	0x00400000	0x00000000	syscall	10: syscall
	0x00400010	0x24110000	addiu \$17,\$0,...	12: li \$s1, 0
	0x00400014	0x3c011001	lui \$1,0x1001	13: la \$s0, list
	0x00400018	0x34300010	ori \$16,\$1,0x...	
	0x0040001c	0x24080003	addiu \$8,\$0,0...	14: li \$t0, 3
	0x00400020	0x11000004	beq \$8,\$0,0x0004	16: beq \$t0, \$zero, loo...
	0x00400024	0x8e090000	lw \$9,0x0000(...	17: lw \$t1, (\$s0)
	0x00400028	0x02298820	add \$17,\$17,\$9	18: add \$s1, \$s1, \$t1
	0x0040002c	0x2108fffe	addi \$8,\$8,0x...	19: addi \$t0, \$t0, -2
	0x00400030	0x08100008	j 0x00400020	20: j loop

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

- ❑ 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).
- ❑ After a correct solution is found, fix the program code then.

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

0x00400010	0x24110000	addiu \$17,\$0,...	12: li \$s1, 0
0x00400014	0x3c011001	lui \$1,0x1001	13: la \$s0, list
0x00400018	0x34300010	ori \$16,\$1,0x...	
0x0040001c	0x24080003	addiu \$8,\$0,0...	14: li \$t0, 3
0x00400020	0x11000004	beq \$8,\$0,0x0004	16: beq \$t0, \$zero, loo...
0x00400024	0x8e090000	lw \$9,0x0000(...	17: lw \$t1, (\$s0)
0x00400028	0x02298820	add \$17,\$17,\$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 registers window shows the following values:

\$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 Data Segment window shows memory addresses and 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 Mars Messages window shows the following output:

```

2 + 3 + 4 = 9
-- program is finished running --
    
```


- ❑ 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?

- ❑ You have learnt:
 - ❑ how to debug a MIPS program using the debugging features in Mars.