

BÁO CÁO THỰC HÀNH LAB02

Bài 1:

The screenshot shows a debugger interface with three main panes:

- Text Segment:** Displays assembly code. The current instruction is highlighted in yellow: `addi $16,$0,0x00003007`. The assembly code consists of two lines:


```
0x00400000 0x20103007 addi $16,$0,0x00003007 2: addi $0,$zero,0x3007
      0x00400004 0x00008020 add $16,$0,$0 3: add $0,$zero,$0
```
- Registers:** Shows the register state. The `$16` register is highlighted in green, indicating it is the current value being displayed in the memory dump.
- Data Segment:** Displays a memory dump from address `0x10010000` to `0x100101e0`. The `$16` register value (0x00003007) is present at address `0x100101d0`.

*Chạy từng dòng lệnh:

This screenshot shows the state of the debugger after executing the first command (`addi $16,$0,0x00003007`). The assembly code now shows the result of the addition:

```
0x00400000 0x20103007 addi $16,$0,0x00003007 2: addi $0,$zero,0x3007
      0x00400004 0x00008020 add $16,$0,$0 3: add $0,$zero,$0
```

The `$16` register is highlighted in green. The memory dump shows the updated value at `0x100101d0`.

This screenshot shows the state of the debugger after executing both commands. The assembly code is now:

```
0x00400000 0x20103007 addi $16,$0,0x00003007 2: addi $0,$zero,0x3007
      0x00400004 0x00008020 add $16,$0,$0 3: add $0,$zero,$0
```

The `$16` register is highlighted in green. The memory dump shows the final value at `0x100101d0`.

*Sự thay đổi giá trị của thanh ghi:

Sự thay đổi giá trị của thanh ghi \$s0 khi chạy từng lệnh :

0x00000000 → 0x00003007 → 0x00000000

Sự thay đổi giá trị của thanh ghi \$pc khi chạy từng lệnh:

0x00400000 → 0x00400004 → 0x00400008

*So sánh mã máy của các lệnh:

addi \$s0, \$zero, 0x3007

op: 8

rs: \$zero

rt: \$s0

imm: 0x3007

0010 0000 0001 0000 0011 0000 0000 0111 => 0x20103007

add \$s0, \$zero, \$0

op:0

rd: \$s0

rs: \$zero

rt: \$0

shamt: 0

funct:32

0000 0000 0000 0000 1000 0000 0010 0000 => 0x00008020

Name	Number	Value
\$zero	0	0x00000000
\$t0	1	0x00000000
\$t1	2	0x00000000
\$t2	3	0x00000000
\$t3	4	0x00000000
\$t4	5	0x00000000
\$t5	6	0x00000000
\$t6	7	0x00000000
\$t7	8	0x00000000
\$t8	9	0x00000000
\$t9	10	0x00000000
\$t10	11	0x00000000
\$t11	12	0x00000000
\$t12	13	0x00000000
\$t13	14	0x00000000
\$t14	15	0x00000000
\$t15	16	0x00000000
\$t16	17	0x00000000
\$t17	18	0x00000000
\$t18	19	0x00000000
\$t19	20	0x00000000
\$t20	21	0x00000000
\$t21	22	0x00000000
\$t22	23	0x00000000
\$t23	24	0x00000000
\$t24	25	0x00000000
\$t25	26	0x00000000
\$t26	27	0x00000000
\$t27	28	0x00000000
\$t28	29	0x7fffffefffffc
\$t29	30	0x00000000
\$t30	31	0x00000000
\$t31	32	0x00000000
\$t32	33	0x00000000
\$t33	34	0x00000000
\$t34	35	0x00000000
\$t35	36	0x00000000
\$t36	37	0x00000000
\$t37	38	0x00000000
\$t38	39	0x00000000
\$t39	40	0x00000000
\$t40	41	0x00000000
\$t41	42	0x00000000
\$t42	43	0x00000000
\$t43	44	0x00000000
\$t44	45	0x00000000
\$t45	46	0x00000000
\$t46	47	0x00000000
\$t47	48	0x00000000
\$t48	49	0x00000000
\$t49	50	0x00000000
\$t50	51	0x00000000
\$t51	52	0x00000000
\$t52	53	0x00000000
\$t53	54	0x00000000
\$t54	55	0x00000000
\$t55	56	0x00000000
\$t56	57	0x00000000
\$t57	58	0x00000000
\$t58	59	0x00000000
\$t59	60	0x00000000
\$t60	61	0x00000000
\$t61	62	0x00000000
\$t62	63	0x00000000
\$t63	64	0x00000000
\$t64	65	0x00000000
\$t65	66	0x00000000
\$t66	67	0x00000000
\$t67	68	0x00000000
\$t68	69	0x00000000
\$t69	70	0x00000000
\$t70	71	0x00000000
\$t71	72	0x00000000
\$t72	73	0x00000000
\$t73	74	0x00000000
\$t74	75	0x00000000
\$t75	76	0x00000000
\$t76	77	0x00000000
\$t77	78	0x00000000
\$t78	79	0x00000000
\$t79	80	0x00000000
\$t80	81	0x00000000
\$t81	82	0x00000000
\$t82	83	0x00000000
\$t83	84	0x00000000
\$t84	85	0x00000000
\$t85	86	0x00000000
\$t86	87	0x00000000
\$t87	88	0x00000000
\$t88	89	0x00000000
\$t89	90	0x00000000
\$t90	91	0x00000000
\$t91	92	0x00000000
\$t92	93	0x00000000
\$t93	94	0x00000000
\$t94	95	0x00000000
\$t95	96	0x00000000
\$t96	97	0x00000000
\$t97	98	0x00000000
\$t98	99	0x00000000
\$t99	100	0x00000000

*Sửa lại lệnh:

Name	Number	Value
\$zero	0	0x00000000
\$t0	1	0x00000000
\$t1	2	0x00000000
\$t2	3	0x00000000
\$t3	4	0x00000000
\$t4	5	0x00000000
\$t5	6	0x00000000
\$t6	7	0x00000000
\$t7	8	0x00000000
\$t8	9	0x00000000
\$t9	10	0x00000000
\$t10	11	0x00000000
\$t11	12	0x00000000
\$t12	13	0x00000000
\$t13	14	0x00000000
\$t14	15	0x00000000
\$t15	16	0x00000000
\$t16	17	0x00000000
\$t17	18	0x00000000
\$t18	19	0x00000000
\$t19	20	0x00000000
\$t20	21	0x00000000
\$t21	22	0x00000000
\$t22	23	0x00000000
\$t23	24	0x00000000
\$t24	25	0x00000000
\$t25	26	0x00000000
\$t26	27	0x00000000
\$t27	28	0x00000000
\$t28	29	0x7fffffefffffc
\$t29	30	0x00000000
\$t30	31	0x00000000
\$t31	32	0x00000000
\$t32	33	0x00000000
\$t33	34	0x00000000
\$t34	35	0x00000000
\$t35	36	0x00000000
\$t36	37	0x00000000
\$t37	38	0x00000000
\$t38	39	0x00000000
\$t39	40	0x00000000
\$t40	41	0x00000000
\$t41	42	0x00000000
\$t42	43	0x00000000
\$t43	44	0x00000000
\$t44	45	0x00000000
\$t45	46	0x00000000
\$t46	47	0x00000000
\$t47	48	0x00000000
\$t48	49	0x00000000
\$t49	50	0x00000000
\$t50	51	0x00000000
\$t51	52	0x00000000
\$t52	53	0x00000000
\$t53	54	0x00000000
\$t54	55	0x00000000
\$t55	56	0x00000000
\$t56	57	0x00000000
\$t57	58	0x00000000
\$t58	59	0x00000000
\$t59	60	0x00000000
\$t60	61	0x00000000
\$t61	62	0x00000000
\$t62	63	0x00000000
\$t63	64	0x00000000
\$t64	65	0x00000000
\$t65	66	0x00000000
\$t66	67	0x00000000
\$t67	68	0x00000000
\$t68	69	0x00000000
\$t69	70	0x00000000
\$t70	71	0x00000000
\$t71	72	0x00000000
\$t72	73	0x00000000
\$t73	74	0x00000000
\$t74	75	0x00000000
\$t75	76	0x00000000
\$t76	77	0x00000000
\$t77	78	0x00000000
\$t78	79	0x00000000
\$t79	80	0x00000000
\$t80	81	0x00000000
\$t81	82	0x00000000
\$t82	83	0x00000000
\$t83	84	0x00000000
\$t84	85	0x00000000
\$t85	86	0x00000000
\$t86	87	0x00000000
\$t87	88	0x00000000
\$t88	89	0x00000000
\$t89	90	0x00000000
\$t90	91	0x00000000
\$t91	92	0x00000000
\$t92	93	0x00000000
\$t93	94	0x00000000
\$t94	95	0x00000000
\$t95	96	0x00000000
\$t96	97	0x00000000
\$t97	98	0x00000000
\$t98	99	0x00000000
\$t99	100	0x00000000

Name	Number	Value
\$zero	0	0x00000000
\$t0	1	0x00000000
\$t1	2	0x00000000
\$t2	3	0x00000000
\$t3	4	0x00000000
\$t4	5	0x00000000
\$t5	6	0x00000000
\$t6	7	0x00000000
\$t7	8	0x00000000
\$t8	9	0x00000000
\$t9	10	0x00000000
\$t10	11	0x00000000
\$t11	12	0x00000000
\$t12	13	0x00000000
\$t13	14	0x00000000
\$t14	15	0x00000000
\$t15	16	0x00000000
\$t16	17	0x00000000
\$t17	18	0x00000000
\$t18	19	0x00000000
\$t19	20	0x00000000
\$t20	21	0x00000000
\$t21	22	0x00000000
\$t22	23	0x00000000
\$t23	24	0x00000000
\$t24	25	0x00000000
\$t25	26	0x00000000
\$t26	27	0x00000000
\$t27	28	0x00000000
\$t28	29	0x7fffffefffffc
\$t29	30	0x00000000
\$t30	31	0x00000000
\$t31	32	0x00000000
\$t32	33	0x00000000
\$t33	34	0x00000000
\$t34	35	0x00000000
\$t35	36	0x00000000
\$t36	37	0x00000000
\$t37	38	0x00000000
\$t38	39	0x00000000
\$t39	40	0x00000000
\$t40	41	0x00000000
\$t41	42	0x00000000
\$t42	43	0x00000000
\$t43	44	0x00000000
\$t44	45	0x00000000
\$t45	46	0x00000000
\$t46	47	0x00000000
\$t47	48	0x00000000
\$t48	49	0x00000000
\$t49	50	0x00000000
\$t50	51	0x00000000
\$t51	52	0x00000000
\$t52	53	0x00000000
\$t53	54	0x00000000
\$t54	55	0x00000000
\$t55	56	0x00000000
\$t56	57	0x00000000
\$t57	58	0x00000000
\$t58	59	0x00000000
\$t59	60	0x00000000
\$t60	61	0x00000000
\$t61	62	0x00000000
\$t62	63	0x00000000
\$t63	64	0x00000000
\$t64	65	0x00000000
\$t65	66	0x00000000
\$t66	67	0x00000000
\$t67	68	0x00000000
\$t68	69	0x00000000
\$t69	70	0x00000000
\$t70	71	0x00000000
\$t71	72	0x00000000
\$t72	73	0x00000000
\$t73	74	0x00000000
\$t74	75	0x00000000
\$t75	76	0x00000000
\$t76	77	0x00000000
\$t77	78	0x00000000
\$t78	79	0x00000000
\$t79	80	0x00000000
\$t80	81	0x00000000
\$t81	82	0x00000000
\$t82	83	0x00000000
\$t83	84	0x00000000
\$t84	85	0x00000000
\$t85	86	0x00000000
\$t86	87	0x

The screenshot shows the QEMU debugger interface with three main panes:

- Registers**: Shows the CPU register state. The \$r0 register is highlighted in yellow.
- Memory Dump**: A large yellow-highlighted area representing memory starting at address 0x00000020.
- Text Segment**: Displays assembly code from the instruction at address 0x00000020. The first few instructions are:


```

      Bkpt      1  Code      Basic
      0x00000020 0x3c01d101 lui    $1,0x0000210
      0x00000024 0x34210341 ori   $1,$1,0x000003d
      0x00000028 0x000018020 add   $16,$0,$1
      0x0000002c 0x000008020 add   $16,$0,$0
      
```

The screenshot shows the Immunity Debugger interface with three main panes:

- Text Segment**: Displays assembly code. The current instruction is `add $0, $zero, $0` at address `0x00400000`. The assembly pane includes columns for Address, Code, Basic, and Source.
- Registers**: Shows the CPU register state. Registers `$zero` through `$t3`, `$a0` through `$a2`, `$s0` through `$s3`, and `$s0` through `$t7` are listed with their values.
- Data Segment**: Displays memory dump information across various segments like `__text`, `__data`, `__bss`, and `__rodata`. It lists addresses from `0x10010000` to `0x10010169` and their corresponding memory values.

The screenshot shows the Immunity Debugger interface with two main panes. The left pane displays assembly code in the 'Text Segment' tab, showing instructions like addi \$0, \$zero, \$0x2110003d. The right pane displays the 'Registers' tab, showing CPU register values such as rax=0, rbp=0x2110003d, and rsi=0. The bottom navigation bar includes tabs for 'Edit', 'Execute', 'Text Segment', 'Data Segment', 'Registers', 'Coproc 1', and 'Coproc 0'.

Khi sửa lệnh lui, vì hằng số ở đây là 32 bit nên để thực hiện được lệnh addi thì phải tách thành hai lệnh basic là lui và ori.

Bài 2:

Registers Coproc 1 Coproc 0

Name	Number	Value
\$zero	0	0x00000000
\$t1	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	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$t0	16	0x00000000
\$t1	17	0x00000000
\$t2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$t0	26	0x00000000
\$t1	27	0x00000000
\$sp	28	0x10008000
\$fp	29	0x7fffffc0
\$ra	30	0x00000000
pc	31	0x00400000
hi		0x00000000
lo		0x00000000

Registers Coproc 1 Coproc 0

Name	Number	Value
\$zero	0	0x00000000
\$t1	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	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$t8	16	0x00000000
\$t9	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
\$sp	28	0x10008000
\$fp	29	0x7fffffc0
\$ra	30	0x00000000
pc	31	0x00400000
hi		0x00000000
lo		0x00000000

*Chạy từng dòng lệnh và quan sát:

Registers Coproc 1 Coproc 0

Name	Number	Value
\$zero	0	0x00000000
\$t1	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	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$t8	16	0x21100000
\$t9	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
\$sp	28	0x10008000
\$fp	29	0x7fffffc0
\$ra	30	0x00000000
pc	31	0x00400004
hi		0x00000000
lo		0x00000000

The screenshot shows the QEMU debugger interface with three main panes:

- Registers** pane: Shows registers \$zero through \$a3, all set to 0x00000000.
- Text Segment** pane: Shows assembly code starting at address 0x00000000. The first instruction is a lui \$t1, \$16. The second instruction is an ori \$t1, \$0, \$0x0000003d, which is highlighted in yellow.
- Data Segment** pane: Shows memory starting at address 0x00010000. All memory locations are filled with 0x00000000.

*Sự thay đổi của các thanh ghi:

\$s0: 0x00000000 → 0x21100000 → 0x2110003d

\$pc: 0x00400000 → 0x00400004 → 0x00400008

*Quan sát cửa sổ Data Segment:

Các byte đầu tiên ở vùng lệnh trùng với cột Code trong cửa sổ Text Segment.

Bài 3:

The screenshot shows the QEMU debugger interface. The assembly code in the left pane is as follows:

```
.text
li $s0, 0x2110003d
li $s1, 0x2
```

The registers window on the right lists the following register values:

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x21100000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x2110003d
\$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	0x10000000
\$sp	29	0x7ffffefffc
\$fp	30	0x00000000
\$ra	31	0x00000000
\$pc		0x0000000c
\$hi		0x00000000
\$lo		0x00000000

The screenshot shows the Immunity Debugger interface with the following details:

- Registers:** Shows CPU registers (r0 to r31) with their names and values.
- Coproc 1 & Coproc 0:** Shows floating-point registers (st0 to st15) with their names and values.
- Text Segment:** Shows assembly code with addresses, opcodes, and comments. The assembly code includes instructions like `lui \$t0, 0x10000000` and `addiu \$t1, \$t0, 0x10000000`.
- Data Segment:** Shows memory dump with columns for address, value (+0), value (+4), etc.

Hằng số 0x2110003d là 32 bit nên lệnh li thứ nhất để thực hiện được cần phải tách thành 2 lệnh basic là lui và ori.

Bài 4:

The screenshot shows the QEMU debugger interface. The assembly code in the left pane is as follows:

```

Edit Execute
mips1.asm mips2.asm mips3.asm mips4.asm
.text
addi $t1, $zero, 5
addi $t2, $zero, -1

add $s0, $t1, $t1
add $s0, $s0, $t2

```

The registers table in the right pane lists the following register values:

Name	Number	Value
\$zero	0	0x00000000
\$t1	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$s0	4	0x00000000
\$s1	5	0x00000000
\$s2	6	0x00000000
\$s3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$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
\$t7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10000000
\$sp	29	0x7fffffc0
\$fp	30	0x00000000
\$ra	31	0x00000000
\$pc		0x00000000
\$hi		0x00000000
\$lo		0x00000000

The screenshot shows the Immunity Debugger interface with two main panes. The left pane, titled 'Text Segment', displays assembly code for the 'Text' section. It includes columns for Bkpt, Address, Code, and Basic. The right pane, titled 'Registers', shows the CPU register state. Both panes have scroll bars.

*Chạy từng lệnh:

The screenshot shows the Immunity Debugger interface with several windows open:

- Registers**: Shows registers from \$zero to \$lo with their corresponding values.
- Coproc 1**: Shows coprocessor 1 values.
- Coproc 0**: Shows coprocessor 0 values.
- Text Segment**: Displays assembly code with memory addresses, opcodes, and comments. The assembly is as follows:


```

      Bkpt Address Code Basic Source
      0x00400000 0x20090000 addi $9,$0,0x00000005 2: addi $t1,$zero,5
      0x00400004 0x20a1ffff addi $10,$0,0xffffffff 3: addi $t2,$zero,-1
      0x00400008 0x129802 addi $16,$9,$9 5: add $s0,$t1,$t1
      0x0040000c 0x02a08020 add $t6,$16,$10 6: add $s0,$s0,$t2
      
```
- Data Segment**: Shows memory dump details for each byte at the current address.

Text Segment		Registers		Coprocs		
Bkpt	Address	Code	Basic	Name	Number	Value
	0x00400000	0x20090005	add \$9,\$0,0x00000005	\$zero	0	0x00000000
	0x00400004	0x20a1ffff	add \$10,\$0,0xffffffff	\$t1	1	0x00000000
	0x00400008	0x20129829	add \$15,\$9,\$9	\$v0	2	0x00000000
	0x0040000c	0x202e9029	add \$16,\$10,\$10	\$v1	3	0x00000000
				\$w0	4	0x00000000
				\$t2	5	0x00000000
				\$s2	6	0x00000000
				\$s3	7	0x00000000
				\$t0	8	0x00000000
				\$t1	9	0x00000000
				\$t2	10	0xffffffff
				\$t3	11	0x00000000
				\$t4	12	0x00000000
				\$t5	13	0x00000000
				\$t6	14	0x00000000
				\$t7	15	0x00000000
				\$w0	16	0x00000000
				\$t1	17	0x00000000
				\$t2	18	0x00000000
				\$s3	19	0x00000000
				\$s4	20	0x00000000
				\$t5	21	0x00000000
				\$t6	22	0x00000000
				\$t7	23	0x00000000
				\$w0	24	0x00000000
				\$t1	25	0x00000000
				\$t2	26	0x00000000
				\$s1	27	0x00000000
				\$gp	28	0x10000000
				\$sp	29	0x7fffffe1c
				\$fp	30	0x00000000
				\$ra	31	0x00000000
				pc		0x00000000
				hi		0x00000000
				lo		0x00000000

The screenshot shows the QEMU debugger interface with several windows open:

- Registers**: Shows CPU registers (r0-r15, r16, r17, r18, r19, r20) with their names, numbers, and values.
- Coproc 1**: Shows coprocessor 1 registers (hi, lo, pc, ra, sp, fp, sr, gp, sp0, sp1, sp2, sp3, sp4, sp5, sp6, sp7, sp8, sp9, sp10, sp11, sp12, sp13, sp14, sp15).
- Coproc 0**: Shows coprocessor 0 registers (zero, at, v0, v1, v2, v3, v4, v5, v6, v7, v8, v9, v10, v11, v12, v13, v14, v15).
- Text Segment**: Shows assembly code with columns for Bkpt, Address, Code, and Basic. The assembly code includes instructions like addi, add, and add \$0.
- Data Segment**: Shows memory dump with columns for Address, Value (+0), Value (+4), Value (+8), Value (+c), Value (+10), Value (+14), Value (+18), and Value (+1c).

The screenshot shows the QEMU debugger interface with three main panes:

- Registers** pane: Shows CPU registers \$zero through \$t17, each with its name, number, and value.
- Coproc 1** and **Coproc 0** panes: Both are currently empty.
- Data Segment** pane: Shows memory starting at address 0x00000000 with values for bytes 0-15.

Kết quả bằng 9 \rightarrow chạy đúng kết quả.

*Sự thay đổi của các thanh ghi:

\$t1 : 0x00000000 → 0x00000005

\$t2 : 0x00000000 → 0xffffffff

\$s0 : 0x00000000 → 0x0000000a → 0x00000009

*Kiểm nghiệm với khuôn mẫu của kiểu lệnh I:

addi \$9, \$0, 0x00000005

op: 8

rs:\$0

rt:\$9

imm: 0x00000005

0010 0000 0000 1001 0000 0000 0000 0101 => 0x20090005

addi \$10, \$0, 0xffffffff

op: 8

rs: \$0

rt: \$10

imm: 0xffffffff

0010 0000 0000 1010 1111 1111 1111 1111 => 0x200affff

*Kiểm nghiệm với khuôn mẫu của kiểu lệnh R:

add \$16, \$9, \$9

op:0

rs:\$9

rt:\$9

rd: \$16

sh:0

fn:32

0000 0001 0010 1001 1000 0000 0010 0000 => 0x01298020

add \$16, \$16, \$10

op:0

rs: \$16

rt: \$10

rd: \$16

sh:0

fn:32

0000 0010 0000 1010 1000 0000 0010 0000 => 0x020a8020

Bài 5:

The screenshot shows a MIPS assembly debugger interface. On the left, there is an assembly code editor with tabs for various files. The current tab is ".text". The assembly code is:

```
.text
    addi $t1, $zero, 4
    addi $t2, $zero, 5

    mul $s0, $t1, $t2
    mul $s0, $s0, 3

    mflo $s1
```

On the right, there is a register dump window titled "Registers". It lists all 32 general-purpose registers (\$zero through \$t1) along with their corresponding numbers and values. The values are mostly zero, except for \$t1 which contains 15, \$s0 which contains 29, and \$s1 which contains 27.

Name	Number	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	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$t8	17	0x00000000
\$t9	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$t5	21	0x00000000
\$s6	22	0x00000000
\$t7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$t10	26	0x00000000
\$t11	27	0x00000000
\$gp	28	0x10000000
\$sp	29	0x7fffffcf
\$fp	30	0x00000000
\$ra	31	0x00000000
\$pc		0x00000000
hi		0x00000000
lo		0x00000000

The screenshot shows the Immunity Debugger interface with several windows open:

- Registers**: Shows registers \$r0 through \$r15, \$sp, \$bp, \$lr, \$pc, \$r16, \$r17, \$r18, \$r19, \$r20, \$r21, \$r22, \$r23, \$r24, \$r25, \$r26, \$r27, \$r28, \$r29, \$r30, \$r31, \$r32, \$r33, \$r34, \$r35, \$r36, \$r37, \$r38, \$r39, \$r40, \$r41, \$r42, \$r43, \$r44, \$r45, \$r46, \$r47, \$r48, \$r49, \$r50, \$r51, \$r52, \$r53, \$r54, \$r55, \$r56, \$r57, \$r58, \$r59, \$r60, \$r61, \$r62, \$r63, \$r64, \$r65, \$r66, \$r67, \$r68, \$r69, \$r70, \$r71, \$r72, \$r73, \$r74, \$r75, \$r76, \$r77, \$r78, \$r79, \$r80, \$r81, \$r82, \$r83, \$r84, \$r85, \$r86, \$r87, \$r88, \$r89, \$r90, \$r91, \$r92, \$r93, \$r94, \$r95, \$r96, \$r97, \$r98, \$r99, \$r9a, \$r9b, \$r9c, \$r9d, \$r9e, \$r9f, \$r9g, \$r9h, \$r9i, \$r9j, \$r9k, \$r9l, \$r9m, \$r9n, \$r9o, \$r9p, \$r9q, \$r9r, \$r9s, \$r9t, \$r9u, \$r9v, \$r9w, \$r9x, \$r9y, \$r9z, \$r9aa, \$r9ab, \$r9ac, \$r9ad, \$r9ae, \$r9af, \$r9ag, \$r9ah, \$r9ai, \$r9aj, \$r9ak, \$r9al, \$r9am, \$r9an, \$r9ao, \$r9ap, \$r9aq, \$r9ar, \$r9as, \$r9at, \$r9au, \$r9av, \$r9aw, \$r9ax, \$r9ay, \$r9az, \$r9ba, \$r9ca, \$r9da, \$r9ea, \$r9fa, \$r9ga, \$r9ha, \$r9ia, \$r9ja, \$r9ka, \$r9la, \$r9ma, \$r9na, \$r9oa, \$r9pa, \$r9qa, \$r9ra, \$r9sa, \$r9ta, \$r9ua, \$r9va, \$r9wa, \$r9xa, \$r9ya, \$r9za.
- Stack**: Shows memory starting at address 0x00400000, containing assembly code and source code.
- Data Segment**: Shows memory starting at address 0x00400000, containing assembly code and source code.

*Giải thích điều bất thường:

- lệnh mul đầu tiên thực hiện bình thường do là lệnh mul basic (nhân hai biến)
 - lệnh mul thứ 2 không phải là lệnh basic (vì nhân hằng với biến), nên thực hiện biến đổi thành hai lệnh (gán hằng số cho thanh ghi tạm rồi mới thực hiện phép nhân)

* Chạy từng lệnh :

The screenshot shows the QEMU debugger interface with several windows open:

- Registers**: Shows a list of registers with their names, numbers, and values. Most registers are zeroed out.
- Text Segment**: Displays assembly code in the format Address: Code. The code includes instructions like addi, mul, and srl.
- Data Segment**: Shows a memory dump with columns for Address, Value (+0), Value (+4), Value (+8), Value (+12), Value (+16), Value (+20), Value (+24), and Value (+28).

The screenshot shows the Immunity Debugger interface with the following panes:

- Registers**: Shows CPU registers from \$zero to \$t0, each with a value of 0x00000000.
- Coproc 1**: Shows floating-point registers \$f0 to \$f11, each with a value of 0x00000000.
- Coproc 0**: Shows floating-point registers \$r0 to \$r11, each with a value of 0x00000000.
- Text Segment**: Displays assembly code for basic arithmetic operations like addi, add, and mul.
- Data Segment**: Displays memory dump data for the address range 0x00010000 to 0x10001000.
- Bottom Bar**: Includes buttons for file operations (File, Edit, Execute), assembly (Text Segment, Data Segment), memory dump (Memory Dump), registers (Registers, Coproc 1, Coproc 0), and status indicators (Hexadecimal Addresses, Hexadecimal Value, ASCII).

The screenshot shows the Immunity Debugger interface with three main panes:

- Assembly Pane:** Displays assembly code with columns for Blk#, Address, Code, and Basic. The code includes instructions like add, sub, and mul.
- Registers Pane:** Shows the CPU registers (r0 to r15, s0 to s2, f0 to f5) with their current values.
- Data Segment Pane:** Displays memory dump with columns for Address, Value (+0), Value (+4), Value (+8), Value (+C), Value (+10), Value (+14), Value (+18), and Value (+1c).

The screenshot shows the Mars Debugger interface with several windows open:

- Text Segment**: Displays assembly code with columns for Blk#, Address, Code, and Basic.
- Registers**: Shows CPU registers (r0 to r16, pc, hi, lo) and their values.
- Registers Coproc 1**: Shows coprocessor 1 registers (f0-f15) and their values.
- Registers Coproc 0**: Shows coprocessor 0 registers (g0-g15) and their values.
- Data Segment**: Displays memory dump with columns for Address, Value (+0), Value (+4), Value (+8), Value (+c), Value (+10), Value (+14), Value (+18), and Value (+1c).
- Registers**: Shows CPU registers (r0 to r16, pc, hi, lo) and their values.
- Registers Coproc 1**: Shows coprocessor 1 registers (f0-f15) and their values.
- Registers Coproc 0**: Shows coprocessor 0 registers (g0-g15) and their values.
- Mars Messages**: Shows a log of messages.
- Run I/O**: Shows input and output status.

* Sự thay đổi của các thanh ghi:

\$t1 : 0x00000000 → 0x00000004

\$t2: 0x00000000 → 0x00000005

\$lo: 0x00000000 → 0x00000014 → 0x0000003c

\$at: 0x00000000 → 0x00000003

\$s0: 0x00000000 → 0x0000003c

\$s1: 0x00000000 → 0x0000003c

Thanh ghi hi không thay đổi giá trị vì kết quả dưới 32 bit, kết quả ở đây được ghi vào thanh ghi lo.

Bài 6:

The Registers window shows the following register values:

Name	Number	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	0x00000000
\$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	0x10000000
\$sp	29	0x7ffffeffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00000000
hi		0x00000000
lo		0x00000000

The Registers window shows the same register values as above.

The Text Segment window displays the assembly code:

```

    .data
X: .word 5
Y: .word -1
Z: .word

.text
    la $t8, X
    la $t9, Y
    lw $t1, 0($t8)
    lw $t2, 0($t9)

    add $s0, $t1, $t1
    add $s0, $s0, $t2

    la $t7, Z
    sw $s0, 0($t7)

```

The Data Segment window shows the memory dump starting at address 0x10010000. The first few bytes are 0x00000005, 0xffffffff, 0x00000000, ..., 0x00000000.

* Lệnh la được biên dịch bằng cách tách thành 2 lệnh basic là lui và ori

* Ở cửa sổ Data Segment :

Địa chỉ của X = \$t8

Địa chỉ của Y = \$t9

Địa chỉ của Z = \$t7

Tương ứng với hằng số khi biên dịch lệnh la thành mã máy.

The Registers window shows the following register values:

Name	Number	Value
\$zero	0	0x00000000
\$t1	1	0x10010000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000005
\$t2	10	0xffffffff
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x10010008
\$s0	16	0x00000009
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x10010000
\$t9	25	0x10010004
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10000000
\$sp	29	0x7ffffeffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00000000
hi		0x00000000
lo		0x00000000

The Text Segment window shows the assembly code:

```

    .data
X: .word 5
Y: .word -1
Z: .word

.text
    lui $t8, 0x10010001
    ori $t8, $t8, 0x00000000
    lui $t9, 0x10010001
    ori $t9, $t9, 0x00000004
    lw $t1, 0($t8)
    lw $t2, 0($t9)
    add $s0, $t1, $t1
    add $s0, $s0, $t2
    la $t7, Z
    sw $s0, 0($t7)

```

The Labels window shows the following labels:

Label	Address
X	0x10010000
Y	0x10010004
Z	0x10010008

The Text Segment window shows the assembly code:

```

    .data
X: .word 5
Y: .word -1
Z: .word

.text
    lui $t8, 0x10010001
    ori $t8, $t8, 0x00000000
    lui $t9, 0x10010001
    ori $t9, $t9, 0x00000004
    lw $t1, 0($t8)
    lw $t2, 0($t9)
    add $s0, $t1, $t1
    add $s0, $s0, $t2
    la $t7, Z
    sw $s0, 0($t7)

```

The Registers window shows the same register values as above.

The Text Segment window shows the assembly code:

```

    .data
X: .word 5
Y: .word -1
Z: .word

.text
    lui $t8, 0x10010001
    ori $t8, $t8, 0x00000000
    lui $t9, 0x10010001
    ori $t9, $t9, 0x00000004
    lw $t1, 0($t8)
    lw $t2, 0($t9)
    add $s0, $t1, $t1
    add $s0, $s0, $t2
    la $t7, Z
    sw $s0, 0($t7)

```

*Chạy từng dòng lệnh và quan sát:

Name	Number	Value
\$zero	0	0x00000000
\$t0	1	0x10000000
\$t1	2	0x00000000
\$t2	3	0x00000000
\$t3	4	0x00000000
\$t4	5	0x00000000
\$t5	6	0x00000000
\$t6	7	0x00000000
\$t7	8	0x00000000
\$t8	9	0x00000000
\$t9	10	0x00000000
\$t10	11	0x00000000
\$t11	12	0x00000000
\$t12	13	0x00000000
\$t13	14	0x00000000
\$t14	15	0x00000000
\$t15	16	0x00000000
\$t16	17	0x00000000
\$t17	18	0x00000000
\$t18	19	0x00000000
\$t19	20	0x00000000
\$t20	21	0x00000000
\$t21	22	0x00000000
\$t22	23	0x00000000
\$t23	24	0x00000000
\$t24	25	0x00000000
\$t25	26	0x00000000
\$t26	27	0x00000000
\$t27	28	0x00000000
\$t28	29	0x7f11fffc
\$t29	30	0x00000000
\$t30	31	0x00000000
pc		0x00400004
hi		0x00000000
lo		0x00000000

Name	Number	Value
\$zero	0	0x00000000
\$t0	1	0x10000000
\$t1	2	0x00000000
\$t2	3	0x00000000
\$t3	4	0x00000000
\$t4	5	0x00000000
\$t5	6	0x00000000
\$t6	7	0x00000000
\$t7	8	0x00000000
\$t8	9	0x00000000
\$t9	10	0x00000000
\$t10	11	0x00000000
\$t11	12	0x00000000
\$t12	13	0x00000000
\$t13	14	0x00000000
\$t14	15	0x00000000
\$t15	16	0x00000000
\$t16	17	0x00000000
\$t17	18	0x00000000
\$t18	19	0x00000000
\$t19	20	0x00000000
\$t20	21	0x00000000
\$t21	22	0x00000000
\$t22	23	0x00000000
\$t23	24	0x00000000
\$t24	25	0x00000000
\$t25	26	0x00000000
\$t26	27	0x00000000
\$t27	28	0x00000000
\$t28	29	0x7f11fffc
\$t29	30	0x00000000
\$t30	31	0x00000000
pc		0x00400004
hi		0x00000000
lo		0x00000000

Name	Number	Value
\$zero	0	0x00000000
\$t0	1	0x10000000
\$t1	2	0x00000000
\$t2	3	0x00000000
\$t3	4	0x00000000
\$t4	5	0x00000000
\$t5	6	0x00000000
\$t6	7	0x00000000
\$t7	8	0x00000000
\$t8	9	0x00000000
\$t9	10	0x00000000
\$t10	11	0x00000000
\$t11	12	0x00000000
\$t12	13	0x00000000
\$t13	14	0x00000000
\$t14	15	0x00000000
\$t15	16	0x00000000
\$t16	17	0x00000000
\$t17	18	0x00000000
\$t18	19	0x00000000
\$t19	20	0x00000000
\$t20	21	0x00000000
\$t21	22	0x00000000
\$t22	23	0x00000000
\$t23	24	0x10000000
\$t24	25	0x00000000
\$t25	26	0x00000000
\$t26	27	0x00000000
\$t27	28	0x00000000
\$t28	29	0x7f11fffc
\$t29	30	0x00000000
\$t30	31	0x00000000
pc		0x00400004
hi		0x00000000
lo		0x00000000

The screenshot shows the QEMU debugger interface with several windows open:

- Text Segment**: Displays assembly code from address 0x00400000 to 0x0040028. The code includes instructions like lui, la, add, and sw.
- Registers**: Shows the CPU registers (r0-r31, fp, pc) with their names, numbers, and values.
- Labels**: Shows labels and their addresses, such as mips6.asm and various \$z labels.
- Data Segment**: Displays memory dump information for address 0x10010000, including raw hex data and ASCII representation.

The screenshot shows the QEMU debugger interface with several windows open:

- Registers**: Shows registers r0 to r31, all initialized to 0.
- Labels**: Shows labels defined in the mips6.asm file, including \$t0 through \$t9.
- Text Segment**: Shows assembly code with addresses from 0x00000000 to 0x00000028. The code includes instructions like la, add, and sw.
- Data Segment**: Shows memory starting at address 0x10010000 with various values assigned to memory locations.

The figure shows a screenshot of the QEMU debugger interface, specifically the GDB-like monitor. It consists of two main windows side-by-side.

Left Window (mips6 assembly):

- Text Segment:** Shows assembly code with columns for Bkpt, Address, Code, and Basic. The code includes instructions like la, lw, add, and sw.
- Data Segment:** Shows memory dump for address 0x10010000, displaying values from 0x00000000 to 0xffffffff.

Right Window (Registers):

- Registers:** Shows the state of various registers including \$zero, \$at, \$v0, \$v1, \$a0, \$a1, \$t0, \$t1, \$s0, \$s1, \$s2, \$s3, \$t2, \$t3, \$t4, \$t5, \$t6, \$t7, \$t8, \$t9, \$t10, \$t11, \$t12, \$t13, \$t14, \$t15, \$t16, \$t17, \$t18, \$t19, \$t20, \$t21, \$t22, \$t23, \$t24, \$t25, \$t26, \$t27, \$t28, \$t29, \$t30, \$t31, \$pc, \$hi, and \$lo.
- Coproc 1:** Shows coprocessor 1 register values.
- Coproc 0:** Shows coprocessor 0 register values.

The assembly code in both windows is identical, showing the execution of the mips6 assembly program. The registers window shows the initial state of the processor, with many registers containing their default values (e.g., \$zero = 0, \$at = 1).

The screenshot shows the QEMU debugger interface with three main panes:

- Registers** pane: Shows CPU registers (r0 to r31) with their names, numbers, and current values.
- CPU Registers** pane: Shows detailed CPU register information for Coproc 1 and Coproc 0.
- Data Segment** pane: Shows memory dump details for address 0x0010010000 (l.data).

The screenshot shows the following windows:

- Text Segment**: Displays assembly code with columns for Blkpt, Address, Code, Basic, and Source. The source code includes instructions like `la $t8, X`, `la $t9, Y`, `lw $t1, 0($t8)`, `lw $t2, 0($t9)`, `add $t0, $t1, $t2`, and `sw $s0, 0($t7)`.
- Data Segment**: Shows memory starting at address 0x10010000 with various values for offset +0, +4, +8, +c, +10, +14, +18, and +1c.
- Labels**: Lists labels from the assembly code: `zero`, `$t0`, `$t1`, `$t2`, `$t3`, `$t4`, `$t5`, `$t6`, `$t7`, `$t8`, `$t9`, `$t10`, `$t11`, `$t12`, `$t13`, `$t14`, `$t15`, `$t16`, `$t17`, `$t18`, `$t19`, `$t20`, `$t21`, `$t22`, `$t23`, `$t24`, `$t25`, `$t26`, `$t27`, `$t28`, `$t29`, `$t30`, `$t31`, `pc`, `hi`, and `lo`.
- Registers**: Shows the state of registers \$zero through \$t31, PC, and HI/LO.

The screenshot shows the same interface as the first one, but with different memory dump ranges. The Data Segment window now shows memory starting at 0x10010000 up to 0x10011000. The Registers window shows the state of registers \$zero through \$t31, PC, and HI/LO.

*Sự thay đổi giá trị các thanh ghi :

\$at : 0x00000000 → 0x10010000

\$t8 : 0x00000000 → 0x10010000

\$t9 : 0x00000000 → 0x10010004

\$t1 : 0x00000000 → 0x00000005

\$t2 : 0x00000000 → 0xffffffff

\$s0 : 0x00000000 → 0x0000000a → 0x00000009

\$t7 : 0x00000000 → 0x10010008

- Vai trò của lệnh lw và sw :

lw \$rt, imm(\$rs) : gán giá trị của thanh ghi \$rs vào thanh ghi \$rt (\$rt = M[\$rs+imm])

ở đây lw gán \$t1 = \$t8 và \$t2 = \$t9

sw \$rt, imm(\$rs) : gán giá trị của thanh ghi \$rt vào thanh ghi \$rs (M[\$rs+imm] = \$rt)

- Các lệnh lb, sb:

lb : chép 1 byte tại vị trí trong bộ nhớ RAM vào byte thấp của thanh ghi.

sb : lưu một byte thấp trong thanh ghi vào vị trí trong bộ nhớ RAM.