

BÁO CÁO THỰC HÀNH TUẦN 3
Lương Thị Tâm
20194663

Bài 1:

-TH1: Cộng hai số trái dấu

***Code:**

```
mips1.asm
1 .data
2 X: .word 0x7fffffff
3 Y: .word 0x80000000
4 .text
5   lw $s1,X
6   lw $s2,Y
7 start:
8   li $t0, 0 # trạng thái ban đầu không có tràn
9   addu $s3, $s1, $s2 # s3=s1+s2
10  xor $t1,$s1,$s2 # kiểm tra dấu của s1 và s2, nếu s1 và s2 cùng dấu t1>0, nếu s1 và s2 khác dấu t1 < 0
11
12  bltz $t1, EXIT # nếu t1 < 0, không tràn, kết thúc kiểm tra
13  slt $t2, $s3, $s1 # t2 = s3 < s1 ? 1 : 0
14  bltz $s1, NEGATIVE #nếu s1 < 0 , nhảy đến NEGATIVE
15  beq $t2, $zero, EXIT # nếu t2 = 0 -> s3>s1 -> không tràn -> kết thúc kiểm tra
16  j OVERFLOW
17 NEGATIVE:
18  bne $t2, $zero, EXIT # nếu t2 = 1 -> s3<s1 -> không tràn -> kết thúc kiểm tra
19 OVERFLOW:
20  li $t0, 1 # có tràn số -> t0 = 1
21 EXIT:
22
```

***Kết quả chạy:**

Text Segment		Labels		Registers		Stack		
Bkpt	Address	Code	Basic	Label	Address	Name	Number	Value
0x00400000	0x9c011000	lw \$s1,X		start	0x00400010	\$zero	0	0
0x00400004	0x8c310000	lw \$t0,0				\$t0	1	269500992
0x00400008	0x9c011000	lw \$t1,0				\$t1	2	0
0x0040000c	0x9c320004	lw \$s2,4(\$t0)				\$s0	3	0
0x00400010	0x00000000	addu \$s3,\$s1,\$s2				\$s1	4	0
0x00400014	0x02329921	addu \$s3,\$s1,\$s2				\$s2	5	0
0x00400018	0x02324826	lui \$t1,19				\$t0	6	0
0x00400020	0x05020006	bltz \$t1,0				\$t1	7	0
0x00400024	0x02324826	lui \$t1,19				\$t2	8	0
0x00400028	0x05020006	bltz \$t1,0				\$t3	9	-1
0x00400032	0x01140003	slt \$t2,\$s3,\$s1				\$t4	10	0
0x00400036	0x01140003	slt \$t2,\$s3,\$s1				\$t5	11	0
0x00400040	0x01140003	slt \$t2,\$s3,\$s1				\$t6	12	0
0x00400044	0x00400034	beq \$t2,\$zero,OVERFLOW				\$t7	13	0
0x00400048	0x01540000	bne \$t2,\$zero,EXIT				\$t8	14	0
0x00400052	0x00400034	bne \$t2,\$zero,EXIT				\$t9	15	0
0x00400056	0x02408001	addiu \$s3,1				\$t10	16	0
0x00400060	0x02408001	addiu \$s3,1				\$t11	17	2147483647
0x00400064	0x02408001	addiu \$s3,1				\$t12	18	-2147483648
0x00400068	0x00000000	li \$t0,1				\$t13	19	-1
0x00400072	0x00000000	li \$t0,1				\$t14	20	0
0x00400076	0x00000000	li \$t0,1				\$t15	21	0
0x00400080	0x00000000	li \$t0,1				\$t16	22	0
0x00400084	0x00000000	li \$t0,1				\$t17	23	0
0x00400088	0x00000000	li \$t0,1				\$t18	24	0
0x00400092	0x00000000	li \$t0,1				\$t19	25	0
0x00400096	0x00000000	li \$t0,1				\$t20	26	0
0x004000a0	0x00000000	li \$t0,1				\$t21	27	0
0x004000a4	0x00000000	li \$t0,1				\$fp	28	269500992
0x004000a8	0x00000000	li \$t0,1				\$sp	29	2147483647
0x004000b2	0x00000000	li \$t0,1				\$t22	30	0
0x004000b6	0x00000000	li \$t0,1				\$ra	31	0
0x004000c0	0x00000000	li \$t0,1				pc		4194360
0x004000c4	0x00000000	li \$t0,1				hi		0
0x004000c8	0x00000000	li \$t0,1				lo		0

-TH2: Cộng hai số dương có tràn số

*Code:

```

1 .data
2 X: .word 0x7fffffff
3 Y: .word 1
4 .text
5     lw $s1,X
6     lw $s2,Y
7 start:
8     li $t0, 0 # trạng thái ban đầu không có tràn
9     addu $s3, $s1, $s2 # s3=s1+s2
10    xor $t1,$s1,$s2 # kiểm tra dấu của s1 và s2, nếu s1 và s2 cùng dấu t1>0, nếu s1 và s2 khác dấu t1 < 0
11
12    bltz $t1, EXIT # nếu t1 < 0, không tràn, kết thúc kiểm tra
13    slt $t2, $s3, $s1 # t2 = s3 < s1 ? 1 : 0
14    bltz $s1, NEGATIVE #nếu s1 < 0 , nhảy đến NEGATIVE
15    beq $t2, $zero, EXIT # nếu t2 = 0 -> s3>s1 -> không tràn -> kết thúc kiểm tra
16    j OVERFLOW
17 NEGATIVE:
18    bne $t2, $zero, EXIT # nếu t2 = 1 -> s3<s1 -> không tràn -> kết thúc kiểm tra
19 OVERFLOW:
20    li $t0, 1 # có tràn số -> t0 = 1
21 EXIT:
22

```

*Kết quả chạy:

Text Segment		Labels		Data Segment	
Blkt	Address	Label	Address	Address	Value
0x00400000	0x3c011001	lui	0x00400000	0x10000000	0
0x00400004	0x48c30000	lw	0x00400000	0x10000000	1
0x00400008	0x3c011000	lui	0x00400004	0x10000000	2
0x0040000c	0x48c30001	lw	0x00400004	0x10000000	3
0x00400010	0x48c30004	lw	0x00400004	0x10000000	4
0x00400014	0x48c30001	lw	0x00400004	0x10000000	5
0x00400018	0x48c30000	lw	0x00400004	0x10000000	6
0x0040001c	0x48c30001	lw	0x00400004	0x10000000	7
0x00400020	0x48c30000	lw	0x00400004	0x10000000	8
0x00400024	0x48c30001	lw	0x00400004	0x10000000	9
0x00400028	0x48c30000	lw	0x00400004	0x10000000	10
0x0040002c	0x48c30001	lw	0x00400004	0x10000000	11
0x00400030	0x48c30000	lw	0x00400004	0x10000000	12
0x00400034	0x48c30001	lw	0x00400004	0x10000000	13
0x00400038	0x48c30000	lw	0x00400004	0x10000000	14
0x0040003c	0x48c30001	lw	0x00400004	0x10000000	15
0x00400040	0x48c30000	lw	0x00400004	0x10000000	16
0x00400044	0x48c30001	lw	0x00400004	0x10000000	17
0x00400048	0x48c30000	lw	0x00400004	0x10000000	18
0x0040004c	0x48c30001	lw	0x00400004	0x10000000	19
0x00400050	0x48c30000	lw	0x00400004	0x10000000	20
0x00400054	0x48c30001	lw	0x00400004	0x10000000	21
0x00400058	0x48c30000	lw	0x00400004	0x10000000	22
0x0040005c	0x48c30001	lw	0x00400004	0x10000000	23
0x00400060	0x48c30000	lw	0x00400004	0x10000000	24
0x00400064	0x48c30001	lw	0x00400004	0x10000000	25
0x00400068	0x48c30000	lw	0x00400004	0x10000000	26
0x00400072	0x48c30001	lw	0x00400004	0x10000000	27
0x00400076	0x48c30000	lw	0x00400004	0x10000000	28
0x00400080	0x48c30001	lw	0x00400004	0x10000000	29
0x00400084	0x48c30000	lw	0x00400004	0x10000000	30
0x00400088	0x48c30001	lw	0x00400004	0x10000000	31
0x00400092	0x48c30000	lw	0x00400004	0x10000000	32
0x00400096	0x48c30001	lw	0x00400004	0x10000000	33
0x004000a0	0x48c30000	lw	0x00400004	0x10000000	34
0x004000a4	0x48c30001	lw	0x00400004	0x10000000	35
0x004000a8	0x48c30000	lw	0x00400004	0x10000000	36
0x004000b2	0x48c30001	lw	0x00400004	0x10000000	37
0x004000b6	0x48c30000	lw	0x00400004	0x10000000	38
0x004000b0	0x48c30001	lw	0x00400004	0x10000000	39
0x004000b4	0x48c30000	lw	0x00400004	0x10000000	40
0x004000b8	0x48c30001	lw	0x00400004	0x10000000	41
0x004000c2	0x48c30000	lw	0x00400004	0x10000000	42
0x004000c6	0x48c30001	lw	0x00400004	0x10000000	43
0x004000c0	0x48c30000	lw	0x00400004	0x10000000	44
0x004000c4	0x48c30001	lw	0x00400004	0x10000000	45
0x004000c8	0x48c30000	lw	0x00400004	0x10000000	46
0x004000cc	0x48c30001	lw	0x00400004	0x10000000	47
0x004000d0	0x48c30000	lw	0x00400004	0x10000000	48
0x004000d4	0x48c30001	lw	0x00400004	0x10000000	49
0x004000d8	0x48c30000	lw	0x00400004	0x10000000	50
0x004000dc	0x48c30001	lw	0x00400004	0x10000000	51
0x004000e0	0x48c30000	lw	0x00400004	0x10000000	52
0x004000e4	0x48c30001	lw	0x00400004	0x10000000	53
0x004000e8	0x48c30000	lw	0x00400004	0x10000000	54
0x004000f2	0x48c30001	lw	0x00400004	0x10000000	55
0x004000f6	0x48c30000	lw	0x00400004	0x10000000	56
0x004000f0	0x48c30001	lw	0x00400004	0x10000000	57
0x004000f4	0x48c30000	lw	0x00400004	0x10000000	58
0x004000f8	0x48c30001	lw	0x00400004	0x10000000	59
0x004000fc	0x48c30000	lw	0x00400004	0x10000000	60
0x00400100	0x48c30001	lw	0x00400004	0x10000000	61
0x00400104	0x48c30000	lw	0x00400004	0x10000000	62
0x00400108	0x48c30001	lw	0x00400004	0x10000000	63
0x0040010c	0x48c30000	lw	0x00400004	0x10000000	64
0x00400110	0x48c30001	lw	0x00400004	0x10000000	65
0x00400114	0x48c30000	lw	0x00400004	0x10000000	66
0x00400118	0x48c30001	lw	0x00400004	0x10000000	67
0x0040011c	0x48c30000	lw	0x00400004	0x10000000	68
0x00400120	0x48c30001	lw	0x00400004	0x10000000	69
0x00400124	0x48c30000	lw	0x00400004	0x10000000	70
0x00400128	0x48c30001	lw	0x00400004	0x10000000	71
0x0040012c	0x48c30000	lw	0x00400004	0x10000000	72
0x00400130	0x48c30001	lw	0x00400004	0x10000000	73
0x00400134	0x48c30000	lw	0x00400004	0x10000000	74
0x00400138	0x48c30001	lw	0x00400004	0x10000000	75
0x0040013c	0x48c30000	lw	0x00400004	0x10000000	76
0x00400140	0x48c30001	lw	0x00400004	0x10000000	77
0x00400144	0x48c30000	lw	0x00400004	0x10000000	78
0x00400148	0x48c30001	lw	0x00400004	0x10000000	79
0x0040014c	0x48c30000	lw	0x00400004	0x10000000	80
0x00400150	0x48c30001	lw	0x00400004	0x10000000	81
0x00400154	0x48c30000	lw	0x00400004	0x10000000	82
0x00400158	0x48c30001	lw	0x00400004	0x10000000	83
0x0040015c	0x48c30000	lw	0x00400004	0x10000000	84
0x00400160	0x48c30001	lw	0x00400004	0x10000000	85
0x00400164	0x48c30000	lw	0x00400004	0x10000000	86
0x00400168	0x48c30001	lw	0x00400004	0x10000000	87
0x0040016c	0x48c30000	lw	0x00400004	0x10000000	88
0x00400170	0x48c30001	lw	0x00400004	0x10000000	89
0x00400174	0x48c30000	lw	0x00400004	0x10000000	90
0x00400178	0x48c30001	lw	0x00400004	0x10000000	91
0x0040017c	0x48c30000	lw	0x00400004	0x10000000	92
0x00400180	0x48c30001	lw	0x00400004	0x10000000	93
0x00400184	0x48c30000	lw	0x00400004	0x10000000	94
0x00400188	0x48c30001	lw	0x00400004	0x10000000	95
0x0040018c	0x48c30000	lw	0x00400004	0x10000000	96
0x00400190	0x48c30001	lw	0x00400004	0x10000000	97
0x00400194	0x48c30000	lw	0x00400004	0x10000000	98
0x00400198	0x48c30001	lw	0x00400004	0x10000000	99
0x0040019c	0x48c30000	lw	0x00400004	0x10000000	100
0x004001a0	0x48c30001	lw	0x00400004	0x10000000	101
0x004001a4	0x48c30000	lw	0x00400004	0x10000000	102
0x004001a8	0x48c30001	lw	0x00400004	0x10000000	103
0x004001ac	0x48c30000	lw	0x00400004	0x10000000	104
0x004001b0	0x48c30001	lw	0x00400004	0x10000000	105
0x004001b4	0x48c30000	lw	0x00400004	0x10000000	106
0x004001b8	0x48c30001	lw	0x00400004	0x10000000	107
0x004001bc	0x48c30000	lw	0x00400004	0x10000000	108
0x004001c0	0x48c30001	lw	0x00400004	0x10000000	109
0x004001c4	0x48c30000	lw	0x00400004	0x10000000	110
0x004001c8	0x48c30001	lw	0x00400004	0x10000000	111
0x004001cc	0x48c30000	lw	0x00400004	0x10000000	112
0x004001d0	0x48c30001	lw	0x00400004	0x10000000	113
0x004001d4	0x48c30000	lw	0x00400004	0x10000000	114
0x004001d8	0x48c30001	lw	0x00400004	0x10000000	115
0x004001dc	0x48c30000	lw	0x00400004	0x10000000	116
0x004001e0	0x48c30001	lw	0x00400004	0x10000000	117
0x004001e4	0x48c30000	lw	0x00400004	0x10000000	118
0x004001e8	0x48c30001	lw	0x00400004	0x10000000	119
0x004001ec	0x48c30000	lw	0x00400004	0x10000000	120
0x004001f0	0x48c30001	lw	0x00400004	0x10000000	121
0x004001f4	0x48c30000	lw	0x00400004	0x10000000	122
0x004001f8	0x48c30001	lw	0x00400004	0x10000000	123
0x004001fc	0x48c30000	lw	0x00400004	0x10000000	124
0x00400200	0x48c30001	lw	0x00400004	0x10000000	125
0x00400204	0x48c30000	lw	0x00400004	0x10000000	126
0x00400208	0x48c30001	lw	0x00400004	0x10000000	127
0x0040020c	0x48c30000	lw	0x00400004	0x10000000	128
0x00400210	0x48c30001	lw	0x00400004	0x10000000	129
0x00400214	0x48c30000	lw	0x00400004	0x10000000	130
0x00400218	0x48c30001	lw	0x00400004	0x10000000	131
0x0040021c	0x48c30000	lw	0x00400004	0x1	

-TH3: Cộng hai số dương không tràn số

*Code:

```

1 .data
2 X: .word 5
3 Y: .word 1
4 .text
5     lw $s1,X
6     lw $s2,Y
7 start:
8     li $t0, 0 # trạng thái ban đầu không có tràn
9     addu $s3, $s1, $s2 # s3=s1+s2
10    xor $t1,$s1,$s2 # kiểm tra dấu của s1 và s2, nếu s1 và s2 cùng dấu t1>0, nếu s1 và s2 khác dấu t1 < 0
11
12    bltz $t1, EXIT # nếu t1 < 0, không tràn, kết thúc kiểm tra
13    slt $t2, $s3, $s1 # t2 = s3 < s1 ? 1 : 0
14    bltz $s1, NEGATIVE #nếu s1 < 0 , nhảy đến NEGATIVE
15    beq $t2, $zero, EXIT # nếu t2 = 0 -> s3>s1 -> không tràn -> kết thúc kiểm tra
16    j OVERFLOW
17 NEGATIVE:
18    bne $t2, $zero, EXIT # nếu t2 = 1 -> s3<s1 -> không tràn -> kết thúc kiểm tra
19 OVERFLOW:
20    li $t0, 1 # có tràn số -> t0 = 1
21 EXIT:
22

```

*Kết quả chạy:

The screenshot shows the assembly code for the MIPS program, the registers, the stack, and a memory dump.

Registers (Registers tab):

Name	Value
\$zero	0
\$at	1
\$vt	2
\$v1	3
\$s0	4
\$s1	5
\$s2	6
\$t0	7
\$t1	8
\$t2	9
\$t3	10
\$t4	11
\$t5	12
\$t6	13
\$t7	14
\$t8	15
\$t9	16
\$t10	17
\$t11	18
\$t12	19
\$t13	20
\$t14	21
\$t15	22
\$t16	23
\$t17	24
\$t18	25
\$t19	26
\$k0	27
\$k1	28
\$gp	29
\$sp	30
\$fp	31
\$ra	4194995
pc	4194995
hi	0
lo	0

Labels (Labels tab):

Name	Address
start	0x00400010
mips1.asm	0x00400010
NEGATIVE	0x00400030
OVERFLOW	0x00400034
EXIT	0x00400038
X	0x10000000
Y	0x10000004

Data Segment (Data Segment tab):

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000000	5	1	0	0	0	0	0	0
0x10010020	0	0	0	0	0	0	0	0
0x10010040	0	0	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0	0	0
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0	0
0x10010100	0	0	0	0	0	0	0	0
0x10010120	0	0	0	0	0	0	0	0
0x10010140	0	0	0	0	0	0	0	0
0x10010160	0	0	0	0	0	0	0	0

Registers (Registers tab):

Name	Value
\$zero	0
\$at	1
\$vt	2
\$v1	3
\$s0	4
\$s1	5
\$s2	6
\$t0	7
\$t1	8
\$t2	9
\$t3	10
\$t4	11
\$t5	12
\$t6	13
\$t7	14
\$t8	15
\$t9	16
\$t10	17
\$t11	18
\$t12	19
\$t13	20
\$t14	21
\$t15	22
\$t16	23
\$t17	24
\$t18	25
\$t19	26
\$k0	27
\$k1	28
\$gp	29
\$sp	30
\$fp	31
\$ra	4194995
pc	4194995
hi	0
lo	0

-TH4: Cộng hai số âm có tràn số

*Code:

```

1 .data
2 X: .word 0x80000000
3 Y: .word -1
4 .text
5     lw $s1,X
6     lw $s2,Y
7 start:
8     li $t0, 0 # trạng thái ban đầu không có tràn
9     addu $s3, $s1, $s2 # s3=s1+s2
10    xor $t1,$s1,$s2 # kiểm tra dấu của s1 và s2, nếu s1 và s2 cùng dấu t1>0, nếu s1 và s2 khác dấu t1 < 0
11
12    bltz $t1, EXIT # nếu t1 < 0, không tràn, kết thúc kiểm tra
13    slt $t2, $s3, $s1 # t2 = s3 < s1 ? 1 : 0
14    bltz $s1, NEGATIVE #nếu s1 < 0 , nhảy đến NEGATIVE
15    beq $t2, $zero, EXIT # nếu t2 = 0 -> s3>s1 -> không tràn -> kết thúc kiểm tra
16    j OVERFLOW
17 NEGATIVE:
18    bne $t2, $zero, EXIT # nếu t2 = 1 -> s3<s1 -> không tràn -> kết thúc kiểm tra
19 OVERFLOW:
20    li $t0, 1 # có tràn số -> t0 = 1
21 EXIT:
22

```

*Kết quả chạy:

Text Segment			Labels			Data Segment		
Bkpt Address	Code	Basic	Label	Address	Name	Number	Value	
0x00400000	0x3c010001 lui \$1,4097	5:	\$at	0x00400092	\$zero	0	0	
0x00400004	0x8c210000 lw \$1,0(1)		\$v0	0x00400093	\$v1	1	0	
0x00400008	0x3c010001 lui \$1,4097	6:	\$v2	0x00400094	\$s0	4	0	
0x0040000c	0x8c200041 lw \$18,4(1)		\$s1	0x00400095	\$s2	5	0	
0x00400010	0x3c010001 lui \$1,4097	9:	\$t0	0x00400096	\$t1	6	0	
0x00400014	0x02298212 addu \$19,\$17,\$18		\$t2	0x00400097	\$t3	7	0	
0x00400018	0x02324626 ker \$9,\$17,\$18	10:	\$t4	0x00400098	\$t10	8	1	
0x00400022	0x052400061lzt \$9,6		\$t5	0x00400099	\$t1	9	2147483647	
0x00400026	0x052400061lzt \$19,17	12:	\$t6	0x0040009a	\$t2	10	0	
0x0040002a	0x062000021lzt \$17,2	14:	\$t7	0x0040009b	\$t3	11	0	
0x0040002e	0x11400003beq \$10,\$0,3	15:	\$t8	0x0040009c	\$t4	12	0	
0x00400032	0x08010000 j 0x00400034	16:	\$t9	0x0040009d	\$t5	13	0	
0x00400036	0x15400001bne \$10,\$0,1	18:	\$t10	0x0040009e	\$t6	14	0	
0x00400040	0x24080001addu \$8,\$0,1	20:	\$t11	0x0040009f	\$t7	15	0	

Data Segment							
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)
0x10010000	-2147483648	-1	0	0	0	0	0
0x10010020	0	0	0	0	0	0	0
0x10010040	0	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0	0
0x100100a0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0
0x10010100	0	0	0	0	0	0	0
0x10010120	0	0	0	0	0	0	0
0x10010140	0	0	0	0	0	0	0
0x10010160	0	0	0	0	0	0	0
0x10010180	0	0	0	0	0	0	0

Labels	Address	Name	Number	Value
start	0x00400015	\$at	1	26950092
NEGATIVE	0x00400030	\$v0	2	0
OVERFLOW	0x00400034	\$s0	3	0
EXIT	0x00400038	\$s1	4	0
X	0x10010000	\$s2	5	0
Y	0x10010004	\$t0	6	0
		\$t1	7	0
		\$t2	8	1
		\$t3	9	2147483647
		\$t4	10	0
		\$t5	11	0
		\$t6	12	0
		\$t7	13	0
		\$t8	14	0
		\$t9	15	0
		\$t10	16	0
		\$t11	17	-2147483648
		\$t12	18	-1
		\$t13	19	2147483647
		\$t14	20	0
		\$t15	21	0
		\$t16	22	0
		\$t17	23	0
		\$t18	24	0
		\$t19	25	0
		\$t20	26	0
		\$t21	27	0
		\$t22	28	26950092
		\$t23	29	2147483647
		\$t24	30	0
		\$ra	31	0
		pc		4194359
		hi		0
		lo		0

-TH5: Cộng hai số âm không tràn số

*Code:

```

1 .data
2 X: .word -10
3 Y: .word -1
4 .text
5     lw $s1,X
6     lw $s2,Y
7 start:
8     li $t0, 0 # trạng thái ban đầu không có tràn
9     addu $s3, $s1, $s2 # s3=s1+s2
10    xor $t1,$s1,$s2 # kiểm tra dấu của s1 và s2, nếu s1 và s2 cùng dấu t1>0, nếu s1 và s2 khác dấu t1 < 0
11
12    bltz $t1, EXIT # nếu t1 < 0, không tràn, kết thúc kiểm tra
13    slt $t2, $s3, $s1 # t2 = s3 < s1 ? 1 : 0
14    bltz $s1, NEGATIVE #nếu s1 < 0 , nhảy đến NEGATIVE
15    beq $t2, $zero, EXIT # nếu t2 = 0 -> s3>s1 -> không tràn -> kết thúc kiểm tra
16    j OVERFLOW
17 NEGATIVE:
18    bne $t2, $zero, EXIT # nếu t2 = 1 -> s3<s1 -> không tràn -> kết thúc kiểm tra
19 OVERFLOW:
20    li $t0, 1 # có tràn số -> t0 = 1
21 EXIT:
22

```

*Kết quả chạy:

The screenshot shows the QEMU debugger interface with three main windows: Text Segment, Data Segment, and Registers.

- Text Segment:** Displays assembly code from mips1.asm. It highlights several instructions in green, such as `addu`, `bltz`, and `beq`.
- Data Segment:** Shows memory dump starting at address 0x10010000. The first few bytes are initialized to -10 and -1 respectively.
- Registers:** Shows the state of various registers. Register \$t0 is set to 0, while \$s1 and \$s2 contain -10 and -1 respectively. Other registers like \$s3, \$t1, \$t2, and \$s0 are also visible.

Bài 2:

a, Extract MSB of\$S0

*Code:

```

1 .data
2 x: .word 0x12345678
3 .text
4     lw $s0, x # $s0 = x
5     andi $t1, $s0, 0xff000000 # $t1 = 0x12000000
6     srl $t1, $t1, 24 # dich 24 bit -> t1 = 0x12
7
8

```

*Kết quả chạy:

The screenshot shows the QEMU debugger interface with three main windows: Text Segment, Data Segment, and Registers.

- Text Segment:** Displays assembly code from mips2.asm. It highlights instructions like `lw`, `andi`, and `srl`.
- Data Segment:** Shows memory dump starting at address 0x10010000. The variable `x` is set to 0x12345678.
- Registers:** Shows the state of registers. \$s0 contains 0x12345678, \$t1 contains 0x12000000 after the `andi` instruction, and \$t1 contains 0x12 after the `srl` instruction.

b, Clear LSB of \$s0

*Code:

```

1 .data
2 x: .word 0x12345678
3 .text
4     lw $s0, x # $s0 = x
5     andi $s0, $s0, 0xfffffffff00 # $s0 = 0x12345600
6
7

```

*Kết quả chạy:

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0xffffffff03
\$v0	2	0x00000000
\$v1	3	0x00000000
\$s0	4	0x00000000
\$t1	5	0x00000000
\$s2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$s2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x12345600
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000

c, Set LSB of \$s0(bits 7 to 0 are set to 1)

*Code:

```

1 .data
2 x: .word 0x12345678
3 .text
4     lw $s0, x # $s0 = x
5     ori $s0, $s0, 0x000000ff # $s0 = 0x123456ff
6
7

```

*Kết quả chạy:

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x10000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$s0	4	0x00000000
\$t1	5	0x00000000
\$s2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$s2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x123456ff
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000

d, Clear \$s0(\$s0=0, must use logical instructions)

*Code:

```

1 .data
2 x: .word 0x12345678
3 .text
4     lw $s0, x # $s0 = x
5     andi $s0, $s0, 0 # $s0 = 0
6
7

```

*Kết quả chạy:

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x10000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$s0	4	0x00000000
\$t1	5	0x00000000
\$s2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$s2	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
\$t10	26	0x00000000
\$t11	27	0x00000000
\$t12	28	0x00000000
\$t13	29	0x7fffffc0
\$t14	30	0x00000000
\$t15	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

Bài 3:

a, abs \$s0,\$s1

*Code:

```

1 # abs $s0, $s1
2 .text
3 addi $s1,$zero -5 # $s1 = -5
4 slt $t0, $s1, $zero # $t0 = $s1 < 0 ? 1 : 0
5 beq $t0, $zero, ABS # $t0 = 0 -> $s1>0 -> nhay den ABS
6 sub $s0, $zero, $s1 # $s0 = -$s1
7 j EXIT
8 ABS:
9 add $s0, $zero, $s1 # $s0 = $s1
10 EXIT:

```

*Kết quả chạy:

The screenshot shows the assembly code for the `mips3_1.asm` file. The code implements the `abs` function. It uses `addi` to set `$s1` to `-5`, then `slt` to determine if `$s1` is less than zero. If true, it branches to the `ABS` label. At the `ABS` label, it performs `sub` to calculate the absolute value and then `j`s to the `EXIT` label. The registers window shows the state of寄存器 (\$zero=0, \$t0=1, \$s0=-5, \$s1=-5). The memory dump shows the stack and heap areas.

Name	Number	Value
\$zero	0	0
\$t0	1	0
\$s0	4	0
\$s1	5	0
\$t1	6	0
\$t2	7	0
\$t3	8	1
\$t4	9	0
\$t5	10	0
\$t6	11	0
\$t7	12	0
\$t8	13	0
\$t9	14	0
\$t10	15	0
\$s2	16	0
\$s1	17	-5
\$s2	18	0
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	0
\$t8	24	0
\$t9	25	0
\$t10	26	0
\$t1	27	0
\$fp	28	288486224
\$t1	29	214747940
\$fp	30	0
\$ra	31	0
pc		4194328
hi		0
lo		0

b, move \$s0, \$s1

*Code:

```

1 # move $s0, $s1
2 .text
3 addi $s1, $zero, 6 # $s1 = 6
4 add $s0, $zero, $s1 # $s0 = $s1

```

*Kết quả chạy:

The screenshot shows the assembly code for the `mips3_1.asm` file. The code implements the `move` function. It adds `6` to `$s1` and then adds `$s1` to `$s0`. The registers window shows the state of寄存器 (\$zero=0, \$t0=1, \$s0=6, \$s1=6). The memory dump shows the stack and heap areas.

Name	Number	Value
\$zero	0	0
\$t0	1	0
\$s1	2	0
\$t1	3	0
\$s0	4	0
\$t2	5	0
\$t3	6	0
\$t4	7	0
\$t5	8	0
\$t6	9	0
\$t7	10	0
\$t8	11	0
\$t9	12	0
\$t10	13	0
\$s2	14	0
\$t1	15	0
\$s0	16	6
\$s1	17	6
\$s2	18	0
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	0
\$t8	24	0
t9	25	0

c, not \$s0, \$s1

*Code:

```

1 #not $s0, $s1
2 .text
3 addi $s1, $zero, 3
4 nor $s0, $s1, $zero
5

```

*Kết quả chạy:

Name	Number	Value
\$zero	0	0x00000000
\$t0	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$s0	4	0x00000000
\$s1	5	0x00000000
\$s2	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	0xfffffffffc
\$s1	17	0x00000003
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000

d, ble \$s1, \$s2, label

*Code:

```

1 #ble $s1, $s2, label
2 .text
3 addi $s1, $zero, 5 # $s1 = 5
4 addi $s2, $zero, 5 # $s2 = 5
5 slt $t0, $s2, $s1 # $t0 = $s2 < $s1 ? 1 : 0
6 beq $t0, $zero, label # $t0=0 -> $s1 <= $s2 nhay den label
7 add $s1, $s1, $s2 # $s1 = $s1 + $s2
8 j exit
9 label:
10 add $s2, $s1, $s2 # $s2 = $s1 + $s2
11 exit:

```

*Kết quả chạy:

Name	Number	Value
\$zero	0	0
\$t1	1	0
\$v0	2	0
\$v1	3	0
\$s0	4	0
\$s1	5	0
\$s2	6	0
\$a3	7	0
\$t0	8	0
\$t1	9	0
\$t2	10	0
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	0
\$s1	17	5
\$s2	18	10
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	0
\$t8	24	0
\$t9	25	0
\$k0	26	0
\$k1	27	0
\$gp	28	26468224
\$sp	29	2147479548
\$t1	30	0
\$ra	31	0
pc		4194332
hi		0
lo		0

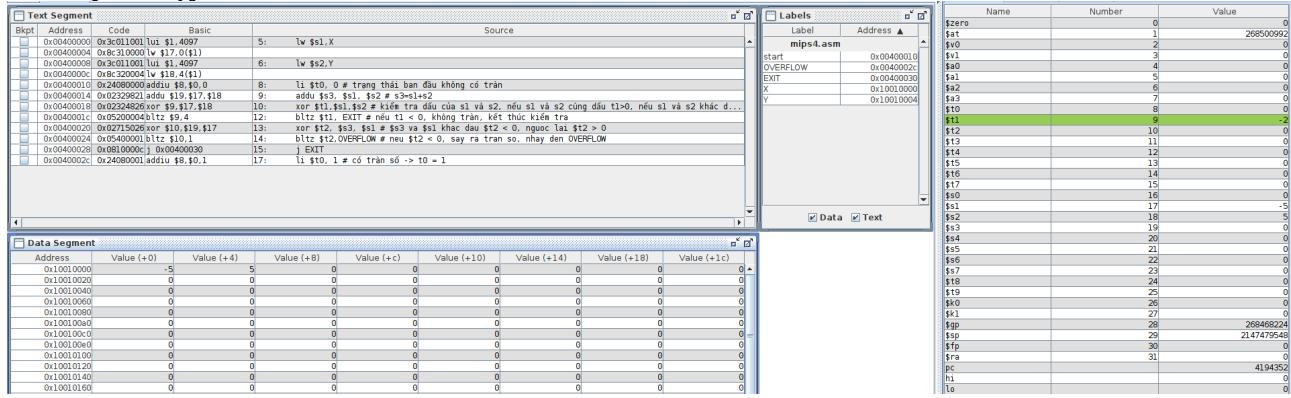
Bài 4:

-TH1: Cộng hai số khác dấu

*Code:

```
1 .data
2 X: .word -5
3 Y: .word 5
4 .text
5     lw $s1,X
6     lw $s2,Y
7 start:
8     li $t0, 0 # trạng thái ban đầu không có tràn
9     addu $s3, $s1, $s2 # s3=s1+s2
10    xor $t1,$s1,$s2 # kiểm tra dấu của s1 và s2, nếu s1 và s2 cùng dấu t1>0, nếu s1 và s2 khác dấu t1 < 0
11
12    bltz $t1, EXIT # nếu t1 < 0, không tràn, kết thúc kiểm tra
13    xor $t2, $s3, $s1 # $s3 và $s1 khác dấu $t2 < 0, ngược lại $t2 > 0
14    bltz $t2,OVERFLOW # neu $t2 < 0, say ra tràn so, nhay den OVERFLOW
15    j EXIT
16 OVERFLOW:
17    li $t0, 1 # có tràn số -> t0 = 1
18 EXIT:
19
```

*Kết quả chạy:



-TH2: Cộng hai số dương không tràn số

***Code:**

```
1 .data
2 X: .word 10
3 Y: .word 5
4 .text
5     lw $s1,X
6     lw $s2,Y
7 start:
8     li $t0, 0 # trạng thái ban đầu không có tràn
9     addu $s3, $s1, $s2 # s3=s1+s2
10    xor $t1,$s1,$s2 # kiểm tra dấu của s1 và s2, nếu s1 và s2 cùng dấu t1>0, nếu s1 và s2 khác dấu t1 < 0
11
12    blitz $t1, EXIT # nếu t1 < 0, không tràn, kết thúc kiểm tra
13    xor $t2, $s3, $s1 # $s3 và $s1 khác dấu $t2 < 0, ngược lại $t2 > 0
14    blitz $t2,OVERFLOW # neu $t2 < 0, say ra tran so, nhay den OVERFLOW
15    j EXIT
16 OVERFLOW:
17    li $t0, 1 # có tràn số -> t0 = 1
18 EXIT:
```

*Kết quả chạy:

-TH3: Cộng hai số dương có tràn số

*Code:

```

1 .data
2 X: .word 0xffffffff
3 Y: .word 5
4 .text
5     lw $s1,X
6     lw $s2,Y
7 start:
8     li $t0, 0 # trạng thái ban đầu không có tràn
9     addu $s3, $s1, $s2 # s3=s1+s2
10    xor $t1,$s1,$s2 # kiểm tra dấu của s1 và s2, nếu s1 và s2 cùng dấu t1>0, nếu s1 và s2 khác dấu t1 < 0
11
12    bltz $t1, EXIT # nếu t1 < 0, không tràn, kết thúc kiểm tra
13    xor $t2, $s3, $s1 # $s3 và $s1 khác dấu $t2 < 0, ngược lại $t2 > 0
14    bltz $t2,OVERFLOW # neu $t2 < 0, say ra tran so, nhay den OVERFLOW
15    j EXIT
16 OVERFLOW:
17    li $t0, 1 # có tràn số -> t0 = 1
18 EXIT:
19

```

*Kết quả chạy:

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

- Text Segment:** Displays assembly code from address 0x00400000 to 0x10010000. It highlights line 17 (li \$t0, 1) in green, indicating it's the current instruction being executed.
- Labels:** A table showing labels, addresses, and their corresponding values. The entry for \$t0 at address 0x10010004 has a value of 1.
- Data Segment:** A table showing memory dump from address 0x10010000 to 0x10011000. The value at address 0x10010004 is shown as 1.

-TH4: Cộng hai số âm không tràn số

*Code:

```

1 .data
2 X: .word -6
3 Y: .word -10
4 .text
5     lw $s1,X
6     lw $s2,Y
7 start:
8     li $t0, 0 # trạng thái ban đầu không có tràn
9     addu $s3, $s1, $s2 # s3=s1+s2
10    xor $t1,$s1,$s2 # kiểm tra dấu của s1 và s2, nếu s1 và s2 cùng dấu t1>0, nếu s1 và s2 khác dấu t1 < 0
11
12    bltz $t1, EXIT # nếu t1 < 0, không tràn, kết thúc kiểm tra
13    xor $t2, $s3, $s1 # $s3 và $s1 khác dấu $t2 < 0, ngược lại $t2 > 0
14    bltz $t2,OVERFLOW # neu $t2 < 0, say ra tràn so, nhay den OVERFLOW
15    j EXIT
16 OVERFLOW:
17    li $t0, 1 # có tràn số -> t0 = 1
18 EXIT:
19

```

*Kết quả chạy:

The screenshot shows the assembly code and its execution state in a debugger. The assembly window displays the MIPS assembly code with comments explaining the logic. The Registers window shows the register values at each step of the execution. The Stack window shows the stack contents. The Memory dump window shows the memory dump starting from address 0x10010000.

Text Segment							
Bkpt	Address	Code	Source	Basic			
	0x00400000	0x3c010000 lui \$1,4097		5:	lw \$s1,X		
	0x00400004	0x3c100000 lw \$t1,0(\$1)					
	0x00400008	0x3c080000 lui \$1,4097		6:	lw \$s2,Y		
	0x0040000c	0x3c180000 lw \$t2,0(\$1)					
	0x00400010	0x24080000 addiu \$s3,\$t0,0		8:	li \$t0, 0 # trạng thái ban đầu không có tràn		
	0x00400014	0x02329821 addu \$s3,\$s1,\$s2 # s3=s1+s2		9:	addu \$s3,\$s1,\$s2 # s3=s1+s2		
	0x00400018	0x02329420 addu \$s3,\$t1,0		10:	xor \$t1,\$s3,\$s1 # kiểm tra dấu của s1 và s2, nếu s1 và s2 cùng dấu t1>0, nếu s1 và s2 khác dấu t1<0		
	0x0040001c	0x02329420 addu \$s3,\$t1,0		11:	bltz \$t1, EXIT # nếu t1 < 0, không tràn, kết thúc kiểm tra		
	0x00400020	0x02715026 xor \$t0,\$t1		12:	xor \$t1,\$s3,\$s1 # s3 và s1 khác dấu t1 < 0, ngược lại t1 > 0		
	0x00400024	0x05400001 bltz \$t0,1		13:	xor \$t2,\$s3,\$s1 # s3 và s1 khác dấu t2 < 0, ngược lại t2 > 0		
	0x00400028	0x08010000 j 0x00400030		14:	bltz \$t2,OVERFLOW # neu \$t2 < 0, say ra tràn so, nhay den OVERFLOW		
	0x0040002c	0x24080001 addiu \$s,\$t0,1		15:	j EXIT		
				17:	li \$t0, 1 # có tràn số -> t0 = 1		

Data Segment								
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	-6	-10	0	0	0	0	0	0
0x10010020	0	0	0	0	0	0	0	0
0x10010040	0	0	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0	0	0
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0	0
0x10010100	0	0	0	0	0	0	0	0
0x10010120	0	0	0	0	0	0	0	0
0x10010140	0	0	0	0	0	0	0	0
0x10010160	0	0	0	0	0	0	0	0
0x10010180	0	0	0	0	0	0	0	0

Labels		Address
Label	Address	
start	0x00400001	
OVERFLOW	0x0040002c	
EXIT	0x00400030	
X	0x10010000	
Y	0x10010004	

Name	Number	Value
zero	0	0
\$t0	1	26850050
\$v0	2	0
\$v1	3	0
\$t0	4	0
\$t1	5	0
\$s2	6	0
\$t2	7	0
\$t0	8	0
\$t1	9	12
\$s2	10	10
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t7	14	0
\$t7	15	0
\$t0	16	0
\$t1	17	-6
\$s2	18	10
\$s3	19	-16
\$t4	20	0
\$t5	21	0
\$s7	22	0
\$t7	23	0
\$t8	24	0
\$t9	25	0
\$t0	26	0
\$t1	27	0
\$sp	28	268468224
\$sp	29	2147479548
\$fp	30	0
\$ra	31	0
gc		4194352
hi		0
lo		0

-TH5: Cộng hai số âm có tròn số

***Code:**

```
1 .data
2 X: .word 0x80000000
3 Y: .word -10
4 .text
5    lw $s1,X
6    lw $s2,Y
7 start:
8    li $t0, 0 # trạng thái ban đầu không có tràn
9    addu $s3, $s1, $s2 # s3=s1+s2
10   xor $t1,$s1,$s2 # kiểm tra dấu của s1 và s2, nếu s1 và s2 cùng dấu t1>0, nếu s1 và s2 khác dấu t1 < 0
11
12   bltz $t1, EXIT # nếu t1 < 0, không tràn, kết thúc kiểm tra
13   xor $t2, $s3, $s1 # $s3 và $s1 khác dấu $t2 < 0, ngược lại $t2 > 0
14   bltz $t2,OVERFLOW # neu $t2 < 0, say ra tran so, nhay den OVERFLOW
15   j EXIT
16 OVERFLOW:
17   li $t0, 1 # có tràn số -> t0 = 1
18 EXIT:
19
```

***Kết quả chạy:**

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

- Assembly Tab:** Displays assembly code for the mips4.asm file. The code includes instructions like lw, addu, xor, and bltz, along with comments and labels such as start, OVERFLOW, EXIT, and various \$r registers.
- Labels Tab:** Shows a list of labels with their corresponding addresses. Labels include start, OVERFLOW, EXIT, and various \$r registers.
- Data Segment Tab:** Displays memory dump information for address 0x10010000. It shows values for offset 0, 4, 8, c, 10, 14, 18, and 1c, all of which are currently 0.

Bài 5:

*Code:

```

1 .data
2 x: .word 6 # x = 6
3 y: .word 8 # y = 8
4 p: .word 0
5 i: .word 0 # i = 0
6 .text
7     lw $s1, x # $s1 = x
8     lw $s2, y # $s2 = y
9     lw $s3, i # $s3 = i
10    la $t8, p
11    li $t0, 1
12 loop:
13     beq $s2, $t0, enloop # y = 1, ket thuc vong lap
14     srl $s2, $s2, 1 # y = y : 2
15     addi $s3, $s3, 1 # i = i + 1
16     j loop
17 enloop:
18     sllv $t1, $s1, $s3 # p = x*y
19     sw $t1, 0($t8)

```

***Kết quả chay:**

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

- Text Segment**: Shows assembly code for mips5.asm. The code includes instructions like lw, addu, beq, srl, add, sll, and sw.
- Registers**: Shows the CPU registers (zero, \$t0-\$t9, \$a0-\$a3, \$s0-\$s3, \$t10-\$t19, \$s4-\$s6, \$t20-\$t27, \$gp, \$sp, \$ra, pc, hi, lo) with their current values.
- Labels**: Shows the labels defined in the assembly code: loop, enloop, x, y, p, i.
- Data Segment**: Shows memory dump at address 0x10000000, displaying values from 0 to 48.