[MIPS - ARRAYS]

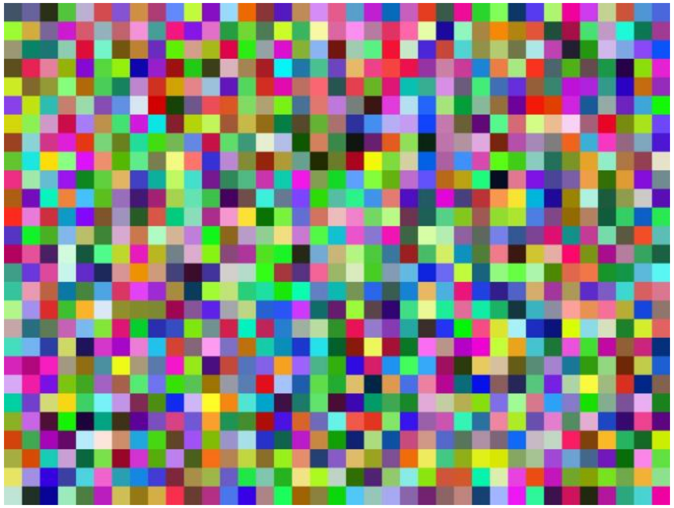
MIPS Assembly Programming

[Arrays]

What is an Array?

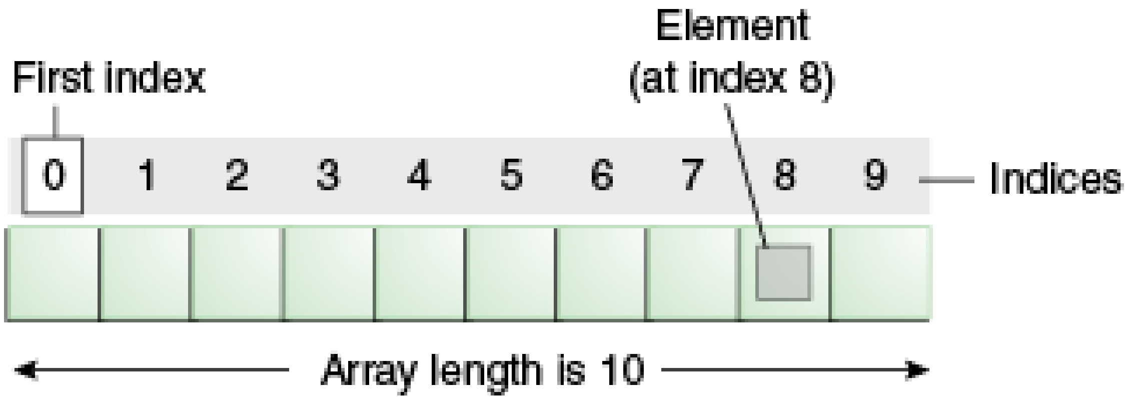
Array

- A list of “items” or “elements”

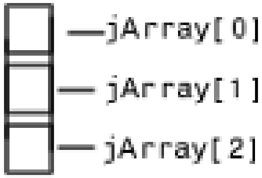


Array

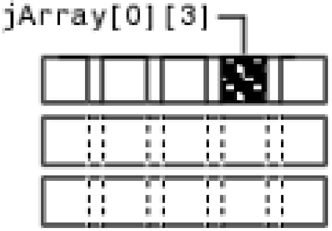
- A list of “items” or “elements”
  - Index: Access each element
  - Size (length): Number of elements.



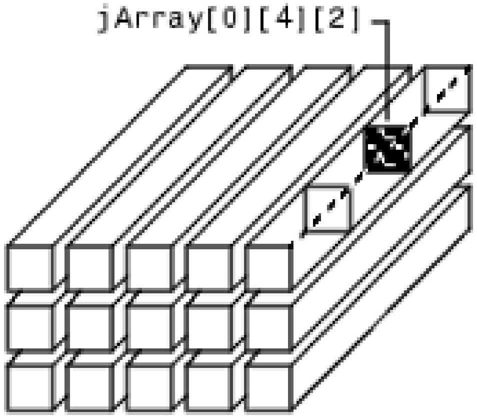
Array Access from Java



Simple Array

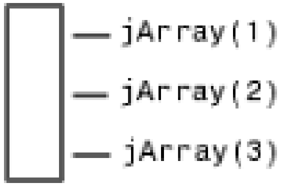


Array of Arrays

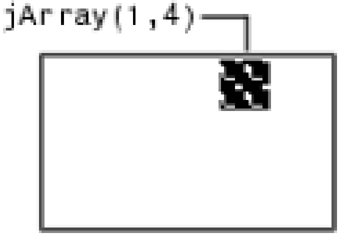


Array of Arrays of Arrays

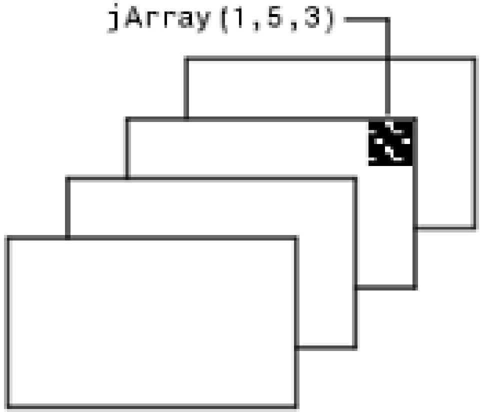
Array Access from MATLAB



One-dimensional Array



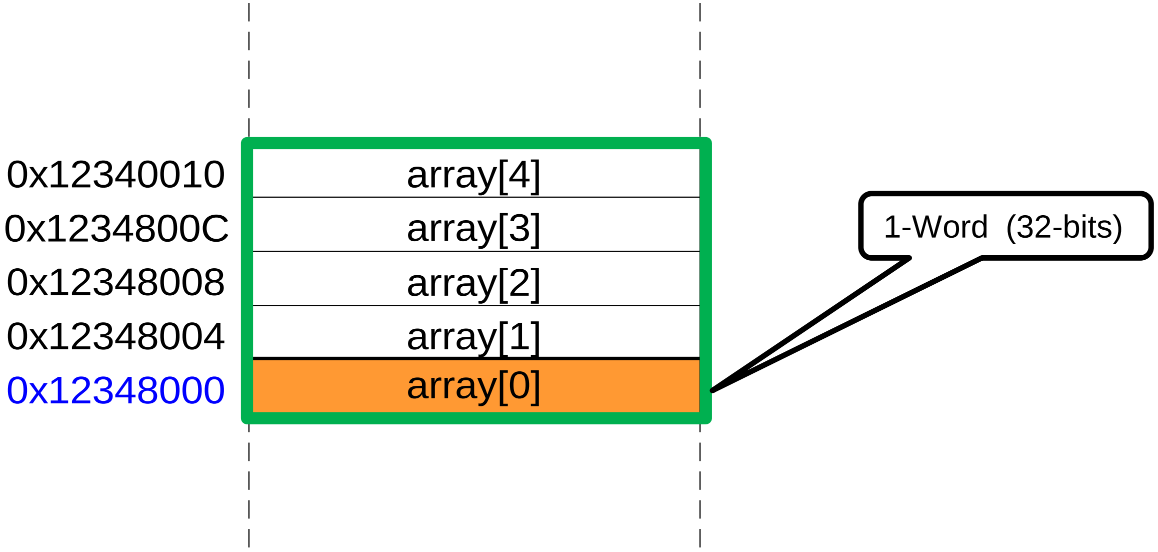
Two-Dimensional Array



Three-Dimensional Array

Array: 5 Elements

- 5-element array
- First element: array[0]
- Base address: 0x12348000
- First step in accessing an array: Load Memory Base Address 0x12348000 … into a Register.



0x12340010	array[4]
0x1234800C	array[3]
0x12348008	array[2]
0x12348004	array[1]
0x12348000	array[0]

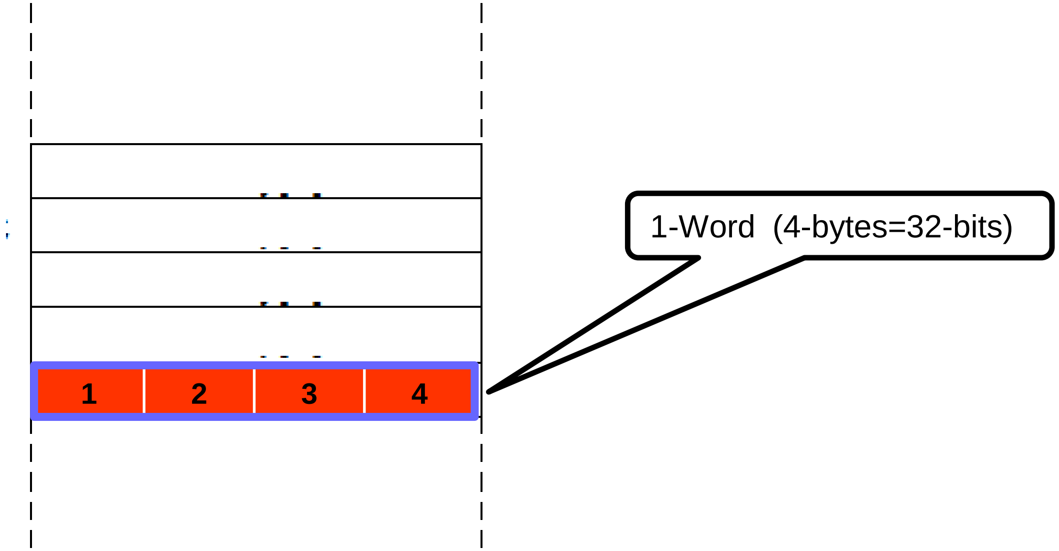
1-Word (32-bits)

Reserve [(4x8)=32 Bits] = 1 Word in RAM

```
        .data
Array:  .space  4
        .text
        la    $t0, Array
```

Reserves a [free] space of 4-bytes or 32-bits (space for just one integer 32 bits = 4-bytes)

Write address of '**Array**' into register **$t0**



In the memory

`0($t0)` ← `5`

Array example with `.space 4`

Example-1

Reserve in memory 4-bytes (1-32 bit word)

Single-Integer Array example

byte-1	byte-2	byte-3	byte-4

```
         .data
Array:   .space   4
         .text
         la     $t0, Array
         li     $t1, 5
         sw     $t1, 0($t0)
         lw     $t2, 0($t0)

         li     $v0, 10
         syscall
```

Reserves a [free] space of 4-bytes or 32-bits (space for just one integer 32 bits = 4-bytes)

Write address of '**Array**' into register **$t0**

$t1 = 5

Store the element (5) to Memory (Array)

Get the element (5) from the Memory (Array)

la $t0, Array; Copy memory address of **Array** into **$t0** (4-bytes or 32-bits)

Resulting values in the registers: $t1, $t2?

```
        .data
Array:  .space  4
        .text
        la      $t0, Array
        li      $t1, 5
        sw      $t1, 0($t0)
        lw      $t2, 0($t0)

        li      $v0, 10
        syscall
```

$t1 = ?

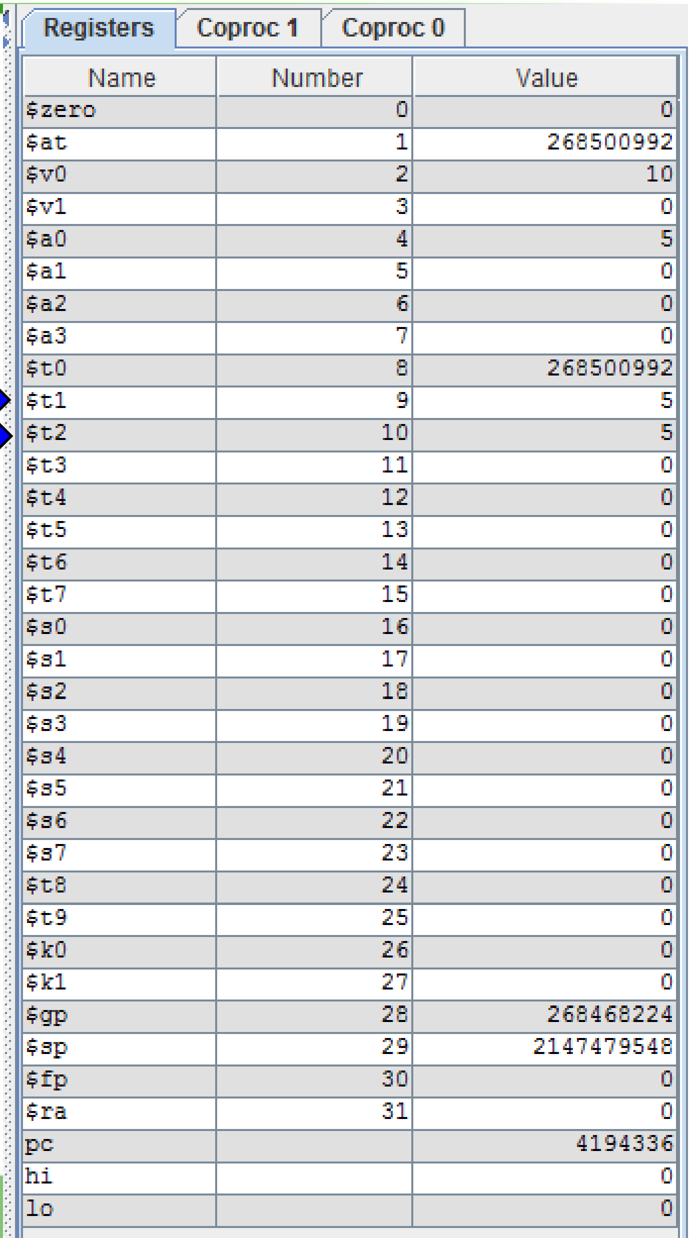
$t2 = ?

Assemble … GO

```
        .data
Array:  .space  4
        .text
        la      $t0, Array
        li      $t1, 5
        sw      $t1, 0($t0)
        lw      $t2, 0($t0)

        li      $v0, 10
        syscall
```

$t1 = 5

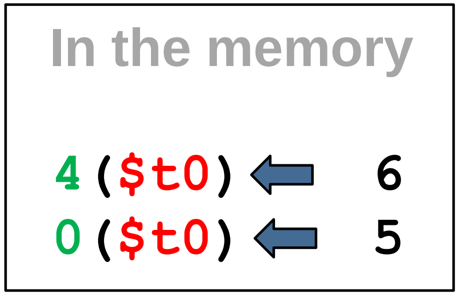
$t2 = 5



Name	Number	Value
$zero	0	0
$at	1	268500992
$v0	2	10
$v1	3	0
$a0	4	5
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	268500992
$t1	9	5
$t2	10	5
$t3	11	0
$t4	12	0
$t5	13	0
$t6	14	0
$t7	15	0
$s0	16	0
$s1	17	0
$s2	18	0
$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	268468224
$sp	29	2147479548
$fp	30	0
$ra	31	0
pc		4194336
hi		0
lo		0

In the memory

`0($t0) ← 5`



In the memory

4($t0) 6

0($t0) 5

2 integer Array example: `.space 8`

Example-2

Reserve in memory 8-bytes (2-32 bit words)

```
         .data
Array:   .space 8
         .text
         #-----------------
         la      $t0, Array
         #-----------------
         li      $t1, 5
         sw      $t1, 0($t0)
         li      $t2, 6
         sw      $t2, 4($t0)
         #-----------------
         lw      $t3, 0($t0)
         lw      $t4, 4($t0)
         #-----------------
         li      $v0, 10
         syscall
```

In the memory: 8-bytes

4($t0)	00000000	00000000	00000000	00000110
0($t0)	00000000	00000000	00000000	00000101

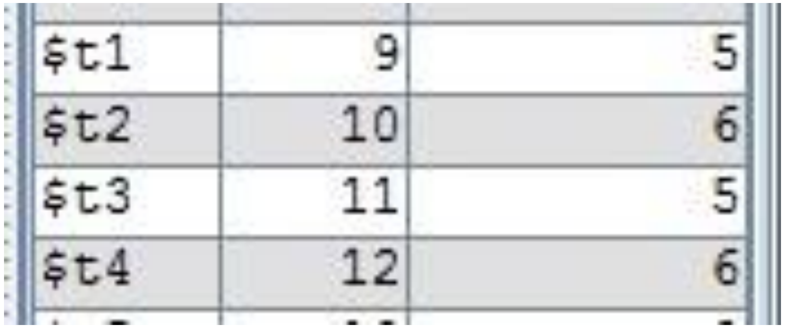
$t1 = ?

$t2 = ?

$t3 = ?

$t4 = ?

```
        .data
Array:  .space 8
        .text
        #-------------------
        la      $t0, Array
        #-------------------
        li      $t1, 5
        sw      $t1, 0($t0)
        li      $t2, 6
        sw      $t2, 4($t0)
        #-------------------
        lw      $t3, 0($t0)
        lw      $t4, 4($t0)
        #-------------------
        li      $v0, 10
        syscall
```



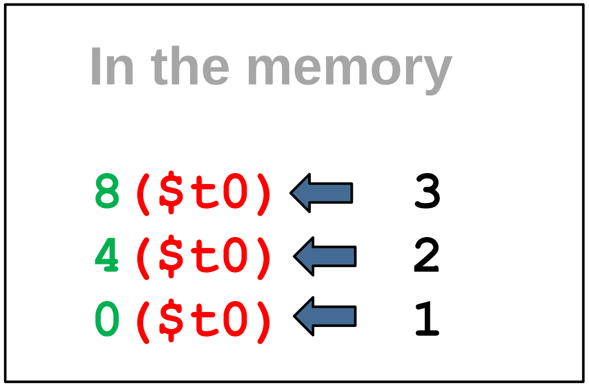
$t1	9	5
$t2	10	6
$t3	11	5
$t4	12	6

$t1 = 5
$t2 = 6
$t3 = 5
$t4 = 6

In the memory

4($t0) ← 6

0($t0) ← 5



In the memory

8($t0) ← 3

4($t0) ← 2

0($t0) ← 1

Array Example with `.word` 1, 2, 3

Example-3

1, 2, 3 is a 3-word (each 32-bit) data array

```
        .text
        .globl main
main:
         la   $t0, nums
         lw   $t1, 0($t0)
         lw   $t2, 4($t0)
         lw   $t3, 8($t0)
         add  $t4, $t1,$t2
         mul  $t5, $t4,$t3
         sw   $t5, 12($t0)
         lw   $t6, 12($t0)
         #-----------------
         li   $v0, 10
         syscall
         #-----------------
         .data
nums:    .word 1, 2, 3
```

In the memory: 12-bytes

8($t0)	00000000	00000000	00000000	00000011
4($t0)	00000000	00000000	00000000	00000010
0($t0)	00000000	00000000	00000000	00000001

$t1 = ?

$t2 = ?

$t3 = ?

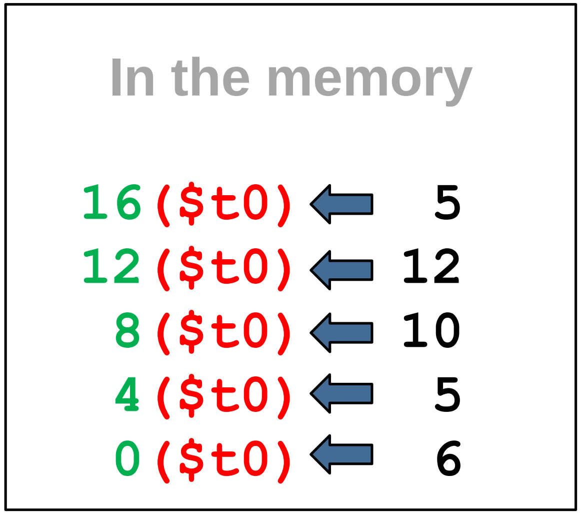
$t4 = ?

$t5 = ?

$t6 = ?

```
     .text
     .globl main
main:
      la  $t0, nums
      lw  $t1, 0($t0)
      lw  $t2, 4($t0)
      lw  $t3, 8($t0)
      add $t4, $t1,$t2
      mul $t5, $t4,$t3
      sw  $t5, 12($t0)
      lw  $t6, 12($t0)
      #----------------
      li  $v0, 10
      syscall
      #----------------
      .data
nums: .word 1, 2, 3
```

```
$t1 = 1
$t2 = 2
$t3 = 3
$t4 = 3
$t5 = 9
$t6 = 9
```



In the memory

16($t0) ← 5
12($t0) ← 12
8($t0) ← 10
4($t0) ← 5
0($t0) ← 6

with:: **`mul`** and **`add`**

Example-4

```
        .data
nums:   .word 6, 5, 10, 12, 5
        .text
        la     $t0, nums
        lw     $t1,  0($t0)
        lw     $t2,  4($t0)
        lw     $t3,  8($t0)
        lw     $t4, 12($t0)
        lw     $t5, 16($t0)
        mul    $t1, $t1, $t1
        mul    $t2, $t2, $t2
        mul    $t3, $t3, $t3
        mul    $t4, $t4, $t4
        mul    $t5, $t5, $t5
        add    $t6, $t1, $t2
        add    $t7, $t6, $t3
        add    $t8, $t7, $t4
        add    $t9, $t8, $t5

        li     $v0, 10
        syscall
```

$t9 = ?

```
       .data
nums:  .word 6, 5, 10, 12, 5
       .text
       la     $t0, nums
       lw     $t1,  0($t0)
       lw     $t2,  4($t0)
       lw     $t3,  8($t0)
       lw     $t4, 12($t0)
       lw     $t5, 16($t0)
       mul    $t1, $t1, $t1
       mul    $t2, $t2, $t2
       mul    $t3, $t3, $t3
       mul    $t4, $t4, $t4
       mul    $t5, $t5, $t5
       add    $t6, $t1, $t2
       add    $t7, $t6, $t3
       add    $t8, $t7, $t4
       add    $t9, $t8, $t5

       li     $v0, 10
       syscall
```

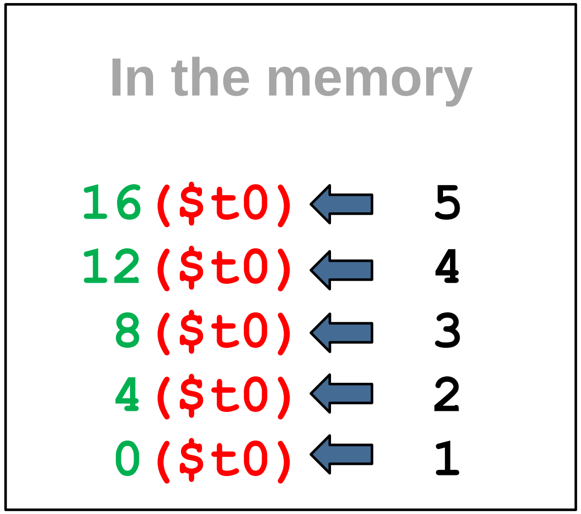
$t9 = 330

What is the implemented function?

```
       .data
nums:  .word 6, 5, 10, 12, 5
       .text
       la    $t0, nums
       lw    $t1,  0($t0)
       lw    $t2,  4($t0)
       lw    $t3,  8($t0)
       lw    $t4, 12($t0)
       lw    $t5, 16($t0)
       mul   $t1, $t1, $t1
       mul   $t2, $t2, $t2
       mul   $t3, $t3, $t3
       mul   $t4, $t4, $t4
       mul   $t5, $t5, $t5
       add   $t6, $t1, $t2
       add   $t7, $t6, $t3
       add   $t8, $t7, $t4
       add   $t9, $t8, $t5

       li    $v0, 10
       syscall
```

$t9 = 330

$$\sum_{j=1}^{5} (\$tj)^2$$



In the memory

16($t0) ← 5

12($t0) ← 4

8($t0) ← 3

4($t0) ← 2

0($t0) ← 1

with:: **`mul`** and **`sll`**

Example-5

```
       .data
nums:  .word 1, 2, 3, 4, 5
       .text

       la $t0, nums
       #-----------------
       lw $t1,  0($t0)
       lw $t2,  4($t0)
       lw $t3,  8($t0)
       lw $t4, 12($t0)
       lw $t5, 16($t0)
       #-----------------
       mul $t6, $t1, $t2
       mul $t6, $t6, $t3
       mul $t6, $t6, $t4
       mul $t6, $t6, $t5
       #-----------------
       sll $t7, $t6, 1
       #-----------------
       li $v0, 10
       syscall
```

$t6 = ?

$t7 = ?

```
      .data
nums: .word 1, 2, 3, 4, 5
      .text

      la $t0, nums
      #----------------
      lw $t1,  0($t0)
      lw $t2,  4($t0)
      lw $t3,  8($t0)
      lw $t4, 12($t0)
      lw $t5, 16($t0)
      #----------------
      mul $t6, $t1, $t2
      mul $t6, $t6, $t3
      mul $t6, $t6, $t4
      mul $t6, $t6, $t5
      #----------------
      sll $t7, $t6, 1
      #----------------
      li $v0, 10
      syscall
```

```
$t6 = 120
$t7 = 240
```

What is the implemented function ?

```
       .data
nums:  .word 1, 2, 3, 4, 5
       .text

       la $t0, nums
       #----------------
       lw $t1,  0($t0)
       lw $t2,  4($t0)
       lw $t3,  8($t0)
       lw $t4, 12($t0)
       lw $t5, 16($t0)
       #----------------
       mul $t6, $t1, $t2
       mul $t6, $t6, $t3
       mul $t6, $t6, $t4
       mul $t6, $t6, $t5
       #----------------
       sll $t7, $t6, 1
       #----------------
       li $v0, 10
       syscall
```

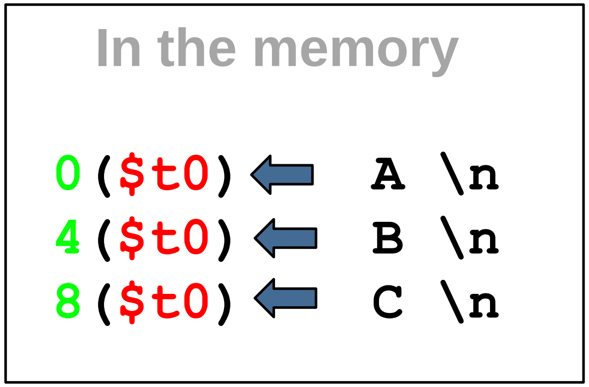
$t6 = 120

$t7 = 240

2(5!)= 120

2($t1*$t2*$t3*$t4*t5)

$$2\prod_{j=1}^{5} \$tj$$



In the memory

```
0($t0) ←  A \n
4($t0) ←  B \n
8($t0) ←  C \n
```

`\n` = new line

Array **character** example with `.space 12`

Example-6

Reserve in memory 12-bytes (3-32 bit words)

```
           .data
let:        .space 12

A:          .asciiz “A   \n”
B:          .asciiz “B   \n”
C:          .asciiz “C   \n”
             .text
            .globl main
main:
            la      $t0, let
            la      $t1, A
            sw      $t1, 0($t0)
            la      $t2, B
            sw      $t2, 4($t0)
            la      $t3, C
            sw      $t3, 8($t0)
            li      $v0, 4
            lw      $a0, 0($t0)
            syscall
            li      $v0, 4
            lw      $a0, 4($t0)
            syscall
            li      $v0, 4
            lw      $a0, 8($t0)
            syscall
            li      $v0, 10
            syscall
```

In the memory: 12 bytes

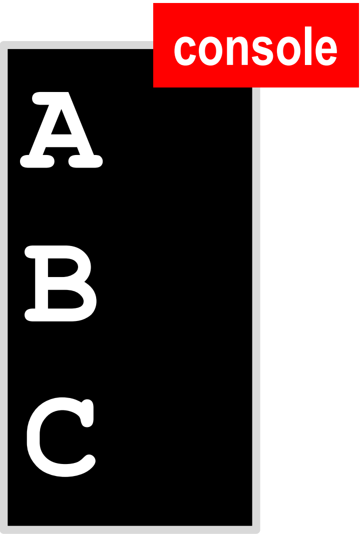
8($t0)	00000000	00000000	00000000	10000011
4($t0)	00000000	00000000	00000000	10000010
0($t0)	00000000	00000000	00000000	10000001

The output?

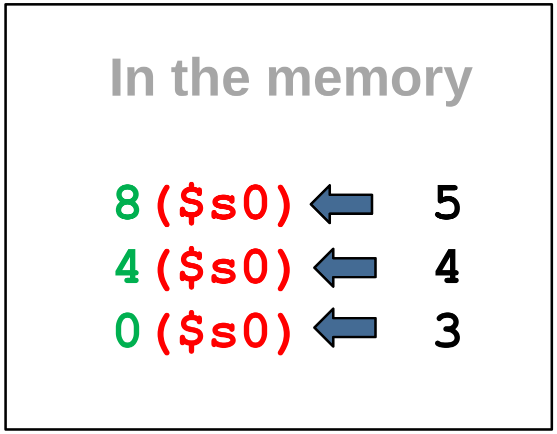
Array with strings of characters

```
        .data
let:    .space 12

A:      .asciiz "A  \n"
B:      .asciiz "B  \n"
C:      .asciiz "C  \n"
         .text
        .globl main
main:
        la      $t0, let
        la      $t1, A
        sw      $t1, 0($t0)
        la      $t2, B
        sw      $t2, 4($t0)
        la      $t3, C
        sw      $t3, 8($t0)
        li      $v0, 4
        lw      $a0, 0($t0)
        syscall
        li      $v0, 4
        lw      $a0, 4($t0)
        syscall
        li      $v0, 4
        lw      $a0, 8($t0)
        syscall
        li      $v0, 10
        syscall
```

The output



```
A
B
C
```



In the memory

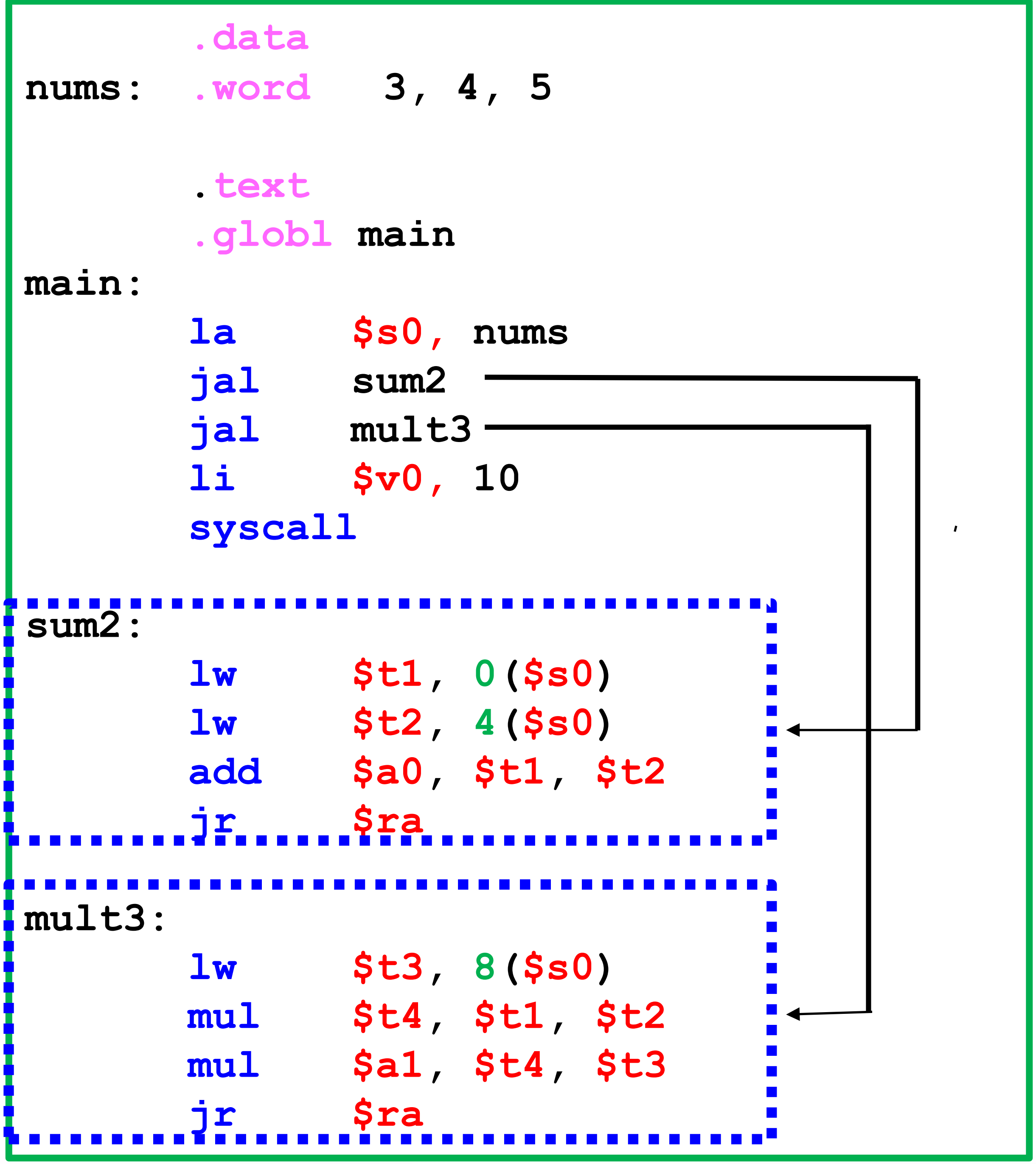
8($s0) ← 5

4($s0) ← 4

0($s0) ← 3

Two Methods and Array

Example-7

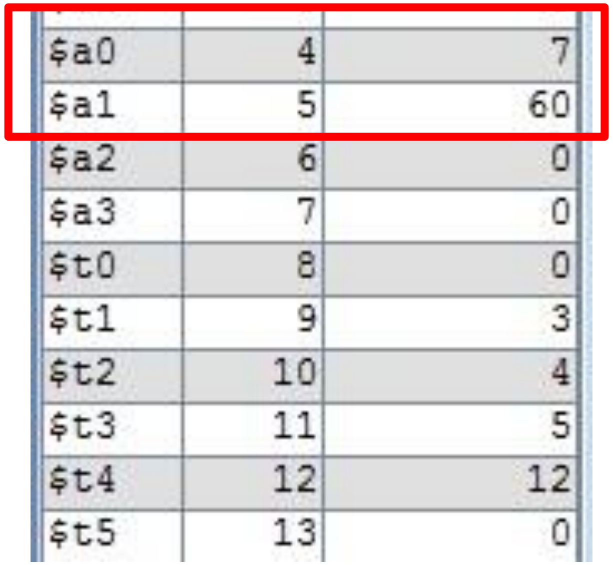


```
        .data
nums:   .word   3, 4, 5

        .text
        .globl main
main:
        la      $s0, nums
        jal     sum2
        jal     mult3
        li      $v0, 10
        syscall

sum2:
        lw      $t1, 0($s0)
        lw      $t2, 4($s0)
        add     $a0, $t1, $t2
        jr      $ra

mult3:
        lw      $t3, 8($s0)
        mul     $t4, $t1, $t2
        mul     $a1, $t4, $t3
        jr      $ra
```

$a0 = ?

$a1 = ?

```
        .data
nums:   .word    3, 4, 5

        .text
        .globl main
main:
        la      $s0, nums
        jal     sum2
        jal     mult3
        li      $v0, 10
        syscall

sum2:
        lw      $t1, 0($s0)
        lw      $t2, 4($s0)
        add     $a0, $t1, $t2
        jr      $ra

mult3:
        lw      $t3, 8($s0)
        mul     $t4, $t1, $t2
        mul     $a1, $t4, $t3
        jr      $ra
```



$a0	4	7
$a1	5	60
$a2	6	0
$a3	7	0
$t0	8	0
$t1	9	3
$t2	10	4
$t3	11	5
$t4	12	12
$t5	13	0

$a0 = 7

$a1 = 60

2 Arrays [4 words (32-bits) each]

Example-8

```
        .text
        .globl main
main:
        la $t0, Array1
        la $t1, Array2

        lw $t2,  0($t0)
        lw $t3,  4($t0)
        lw $t4,  8($t0)
        lw $t5, 12($t0)

        lw $t6,  0($t1)
        lw $t7,  4($t1)
        lw $t8,  8($t1)
        lw $t9, 12($t1)

        sw $t2,  0($t1)
        sw $t3,  4($t1)
        sw $t4,  8($t1)
        sw $t5, 12($t1)

        sw $t6,  0($t0)
        sw $t7,  4($t0)
        sw $t8,  8($t0)
        sw $t9, 12($t0)

        li $v0, 10
        syscall

        .data
Array1: .word 1,  7, 10,  0
Array2: .word 8, 13,  2, 15
```

Array1

Array2

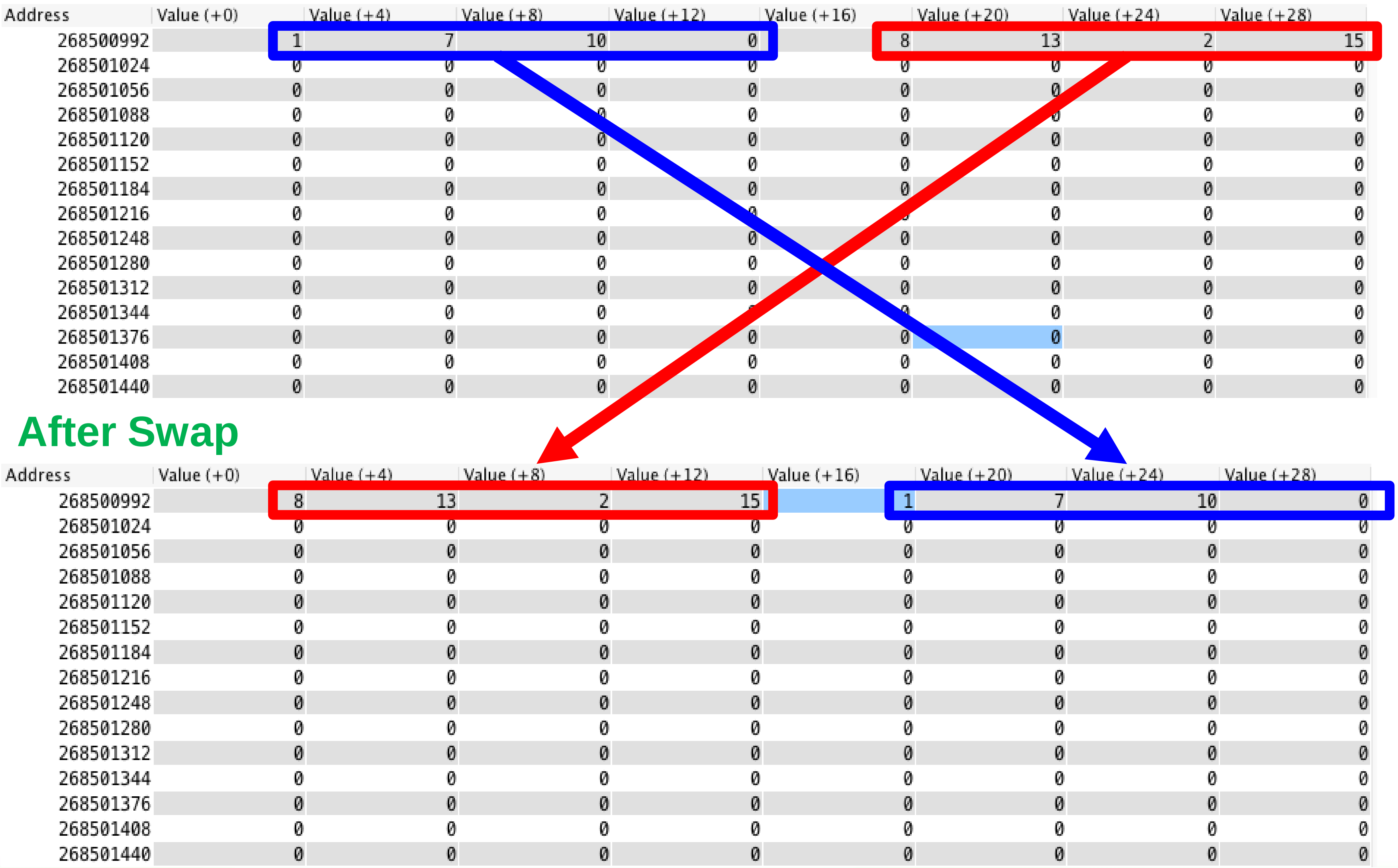
What is the implemented function?

Array Swapping

```
          .text
          .globl main
main:
          la $t0, Array1
          la $t1, Array2

          lw $t2,  0($t0)
          lw $t3,  4($t0)
          lw $t4,  8($t0)
          lw $t5, 12($t0)

          lw $t6,  0($t1)
          lw $t7,  4($t1)
          lw $t8,  8($t1)
          lw $t9, 12($t1)

          sw $t2,  0($t1)
          sw $t3,  4($t1)
          sw $t4,  8($t1)
          sw $t5, 12($t1)

          sw $t6,  0($t0)
          sw $t7,  4($t0)
          sw $t8,  8($t0)
          sw $t9, 12($t0)

          li $v0, 10
          syscall

          .data
Array1:   .word 1,  7, 10,  0
Array2:   .word 8, 13,  2, 15
```

Result (in the memory)

Before Swap



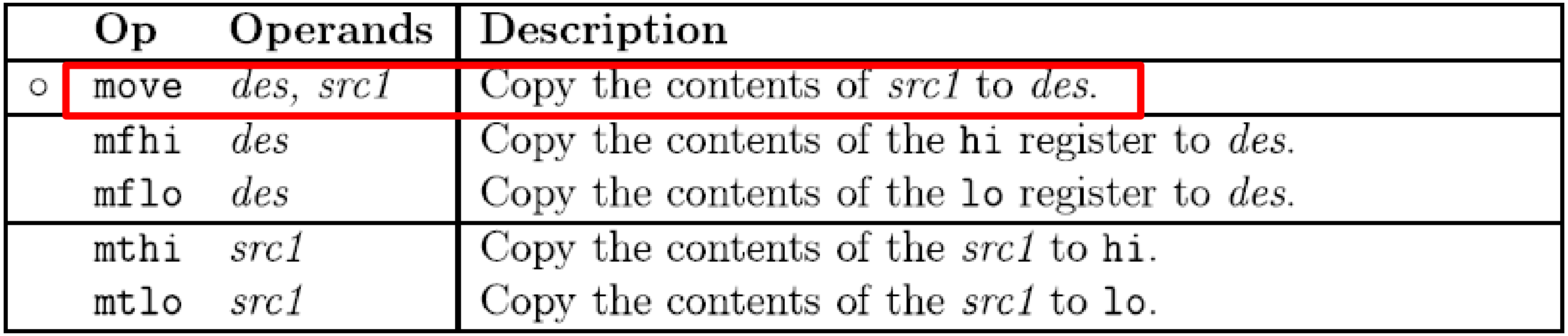
Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268500992	1	7	10	0	8	13	2	15
268501024	0	0	0	0	0	0	0	0
268501056	0	0	0	0	0	0	0	0
268501088	0	0	0	0	0	0	0	0
268501120	0	0	0	0	0	0	0	0
268501152	0	0	0	0	0	0	0	0
268501184	0	0	0	0	0	0	0	0
268501216	0	0	0	0	0	0	0	0
268501248	0	0	0	0	0	0	0	0
268501280	0	0	0	0	0	0	0	0
268501312	0	0	0	0	0	0	0	0
268501344	0	0	0	0	0	0	0	0
268501376	0	0	0	0	0	0	0	0
268501408	0	0	0	0	0	0	0	0
268501440	0	0	0	0	0	0	0	0

After Swap

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268500992	8	13	2	15	1	7	10	0
268501024	0	0	0	0	0	0	0	0
268501056	0	0	0	0	0	0	0	0
268501088	0	0	0	0	0	0	0	0
268501120	0	0	0	0	0	0	0	0
268501152	0	0	0	0	0	0	0	0
268501184	0	0	0	0	0	0	0	0
268501216	0	0	0	0	0	0	0	0
268501248	0	0	0	0	0	0	0	0
268501280	0	0	0	0	0	0	0	0
268501312	0	0	0	0	0	0	0	0
268501344	0	0	0	0	0	0	0	0
268501376	0	0	0	0	0	0	0	0
268501408	0	0	0	0	0	0	0	0
268501440	0	0	0	0	0	0	0	0

More MIPS Instructions

Data Movement

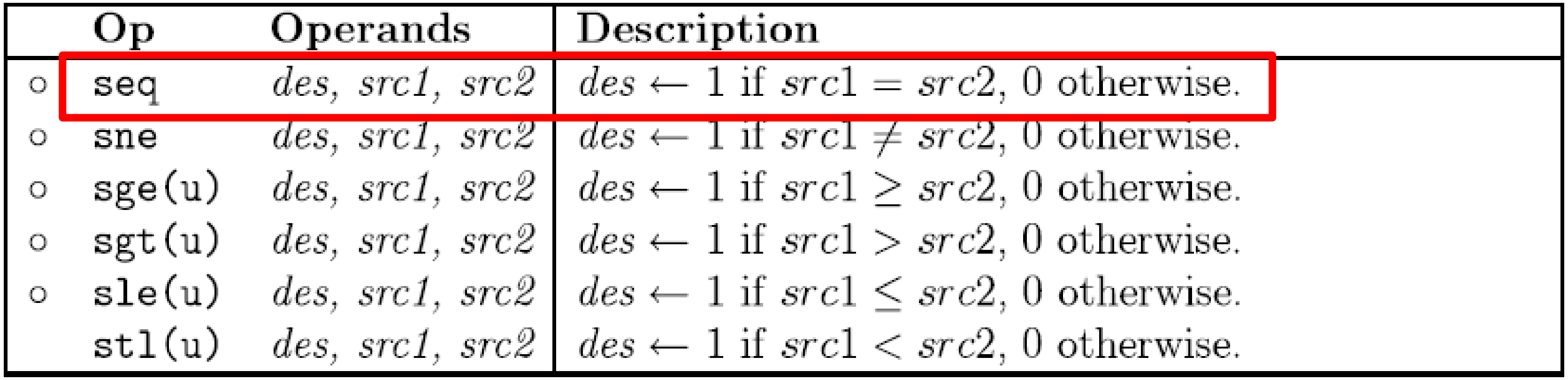


	Op	Operands	Description
○	`move`	*des*, *src1*	Copy the contents of *src1* to *des*.
	`mfhi`	*des*	Copy the contents of the `hi` register to *des*.
	`mflo`	*des*	Copy the contents of the `lo` register to *des*.
	`mthi`	*src1*	Copy the contents of the *src1* to `hi`.
	`mtlo`	*src1*	Copy the contents of the *src1* to `lo`.

mfhi/mflo ➔ **m**ove **f**rom **hi/lo**

mthi/mtlo ➔ **m**ove **t**o **hi/lo**

Comparison, seq



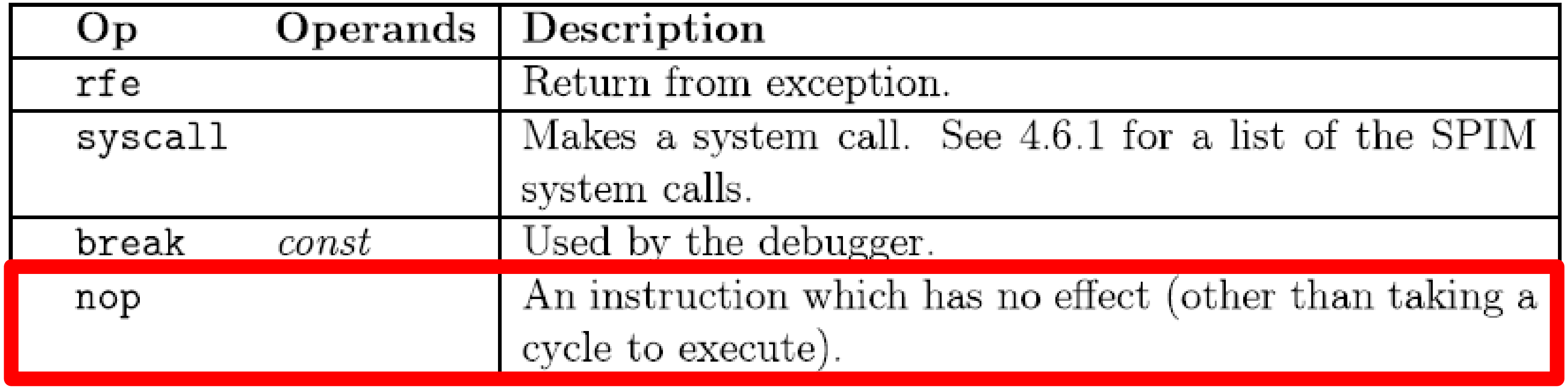
	Op	Operands	Description
○	`seq`	*des, src1, src2*	*des* ← 1 if *src*1 = *src*2, 0 otherwise.
○	`sne`	*des, src1, src2*	*des* ← 1 if *src*1 ≠ *src*2, 0 otherwise.
○	`sge(u)`	*des, src1, src2*	*des* ← 1 if *src*1 ≥ *src*2, 0 otherwise.
○	`sgt(u)`	*des, src1, src2*	*des* ← 1 if *src*1 > *src*2, 0 otherwise.
○	`sle(u)`	*des, src1, src2*	*des* ← 1 if *src*1 ≤ *src*2, 0 otherwise.
	`stl(u)`	*des, src1, src2*	*des* ← 1 if *src*1 < *src*2, 0 otherwise.

seq = des, src1, src2

seq = **s**et re**g**ister **des** to 1 if **src1** = **src2**

`slt $t0, $s1, $s2` (set if less than) # set: `$t0` to 1 if `$s1 < $s2`

Exception Handling, `nop`



Op	Operands	Description
`rfe`		Return from exception.
`syscall`		Makes a system call. See 4.6.1 for a list of the SPIM system calls.
`break`	*const*	Used by the debugger.
`nop`		An instruction which has no effect (other than taking a cycle to execute).

`nop` are used to overcome data-hazards in MIPS pipelined-processors