SIMPLE ARRAY MANIPULATION

1. Array Declaration

Arrays are stored in the Data Segment of a MIPS program. Its elements are accessed via their addresses in memory.

Data Types

Instructions	Syntax	Description
.ascii	str	Store string in memory without null terminator (\ n)
.asciiz	str	Store string in memory with null terminator (\ n)
.byte	b1, b2, … , bn	Store n bytes in memory. Can be written in base 10, or hex. Each value is separated by a comma (,)
.word	h1, h2, … , hn	Store n halfword (2 bytes) in memory. Can be written in base 10, or hex. Each value is separated by a comma (,)
.halfword	w1, w2, … , wn	Store n word (4 bytes) in memory. Can be written in base 10, or hex. Each value is separated by a comma (,)

Few example of array declaration:

```
arr:
.word 1,2,3,4,5 #each entry is allocated 4 bytes

chars:
.byte 'a', 'b', 'c' #each entry is allocated 8 bits

string:
.asciiz "POK is Fun" #String is representated as an array of
characters
```

2. Array Manipulation

Load

Instructions	Syntax	Description
la	$t, label	Put the address of **label** into $t. This can be used to obtain the address of an array (its first element).
lw	$t, i($s)	$t =MEM[$s + i]. Load **word** into a register from the specified address i($s).
li	$t, i($s)	$t =MEM[$s + i]. Load **byte** into a register from the specified address i($s).
lb	$t, immediate	$t = 32 bit immediate value

Store

Instructions	Syntax	Description
sw	$t, i($s)	MEM [$s + i]:4 = $t. Store word from a register ($t) into

		the specified address ($s). This can be used to change the contents of an array

Arithmetic Operation

Instructions	Syntax	Description
add	$t1, $t2, $t3	Set $t1 to ($t2 + $t3)
addi	$t1, $t2, immediate	Set $t1 to ($t2 + immediate)
sub	$t1, $t2, $t3	Set $t1 to ($t2 - $t3)
subi	$t1, $t2, immediate	Set $t1 to ($t2 - immediate)
div	$t1, $t2	Divide $t1 by $t2 then set **hi** to remainder and **lo** to quotinent
div	$t1, $t2, $t3/immediate	Set $t1 to ($t2 divided by $t3/immediate, integer divison)
mult	$t1, $t2	set **hi** to high-order 32 bits and **lo** to low-order 32 bits of the products of $t1 and $t2
mul	$t1, $t2, $t3/immediate	set **hi** to high-order 32 bits, **lo** and $t1 to low-order 32 bits of the products of $t2 and $t3/immediate

Data Movement

Instructions	Syntax	Description
move	$t1, $t2	Set $t1 to content of $t2
mfhi	$t	Set $t to content of hi
mflo	$t	Set $t to content of lo

3. Array Iteration

Branch

Instructions	Syntax	Description
beq	$t0, $t1, target	Branch to target if $t0 = $t1
blt	$t0, $t1, target	Branch to target if $t0 < $t1
ble	$t0, $t1, target	Branch to target if $t0 <= $t1
bgt	$t0, $t1, target	Branch to target if $t0 > $t1
bge	$t0, $t1, target	Branch to target if $t0 >= $t1
bne	$t0, $t1, target	Branch to target if $t0 <> $t1

Jump

Instructions	Syntax	Description
j	target	Unconditional jump to program label target
jr	$t	Jump to address contained in $t. To return to return address, use jr $ra
jal	target	Jump and link. PC+1 (the next instruction) will be copied to register $ra (return address register)

4. Example Code

```
#The following code will calculate the result of list[0] + list[3] +
list[5]

.data
      list: .word 1,2,3,4,5,6,7,8 # array definition s0)

.text
.globl main
main:
      la $t1, list              # put address of list into $t1
      lw $t2, 0($t1)            # get the value of list[0] into $t2
      lw $t3, 12($t1)           # get the value of list[3] into $t3
      add $t1, $t1, 20          # move the pointer to sixth data
      lw $t4, 0($t1)            # get the value of list[5] into $t4
      add $t5, $t3, $t2         # $t5 = $t2 + $t3
      add $t5, $t5, $t4         # $t5 = $t5 + $t4
```

5. Reference

- http://alumni.cs.ucr.edu/~vladimir/cs161/mips.html
- http://people.cs.pitt.edu/~xujie/cs447/AccessingArray.htm