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

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

Here are a few examples of array declaration:

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

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

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

2. Array Manipulation

Load

Instruction	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).
lb	$t, i ($s)	$t = MEM[$s + i]. Load **byte** into a register from the specified address i($s).
li	$t, imm	$t = 32 bit immediate value

Load can be used to retrieve the elements of an array.

Store

Instruction	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.

3. Array Iteration

Branch

Instruction	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
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

Instruction	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

This is an example of MIPS array operation:

```
#The following code will calculate the result of list[1] + list[4]
.globl main
main:
  la $t1, list          # put address of list into $t1
  lw $t2, 0($t1)        # get the value of list[1] into $t2
  lw $t3, 12($t1)        # get the value of list[4] into $t3
  add $t4, $t3, $t2     # $t4 = $t2 + $t3

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

Reference:

http://people.cs.pitt.edu/~xujie/cs447/AccessingArray.htm
http://alumni.cs.ucr.edu/~vladimir/cs161/mips.html
http://logos.cs.uic.edu/366/notes/mips%20quick%20tutorial.htm

Exercise

Yolo is working on a math problem. As he is working with large numbers, he is having a hard time to solve it manually. To make things easier, he wants you to help him create a MIPS program that will solve the problem.

The program is fairly simple. Sadly, Yolo is not familiar with MIPS language. To help you, he has made the program in Python language:

```
myList=[8,10,2983,12,3,7]


total=0
for i in range(len(myList)):
  if myList[i]%2 == 0:
    total+=myList[i]
```

In MIPS, **`total`** will be stored in register **`$s1`**. Now it is your turn to make the MIPS program to help Yolo.

Specification:

- Input:
  - An array of **positive numbers** (don't forget to include/define the array in your code!)
- Output:
  - `total` (stored in **`$s1`**)
- Save the file with format: `[Nama]_[NPM]_[Kelas].asm`