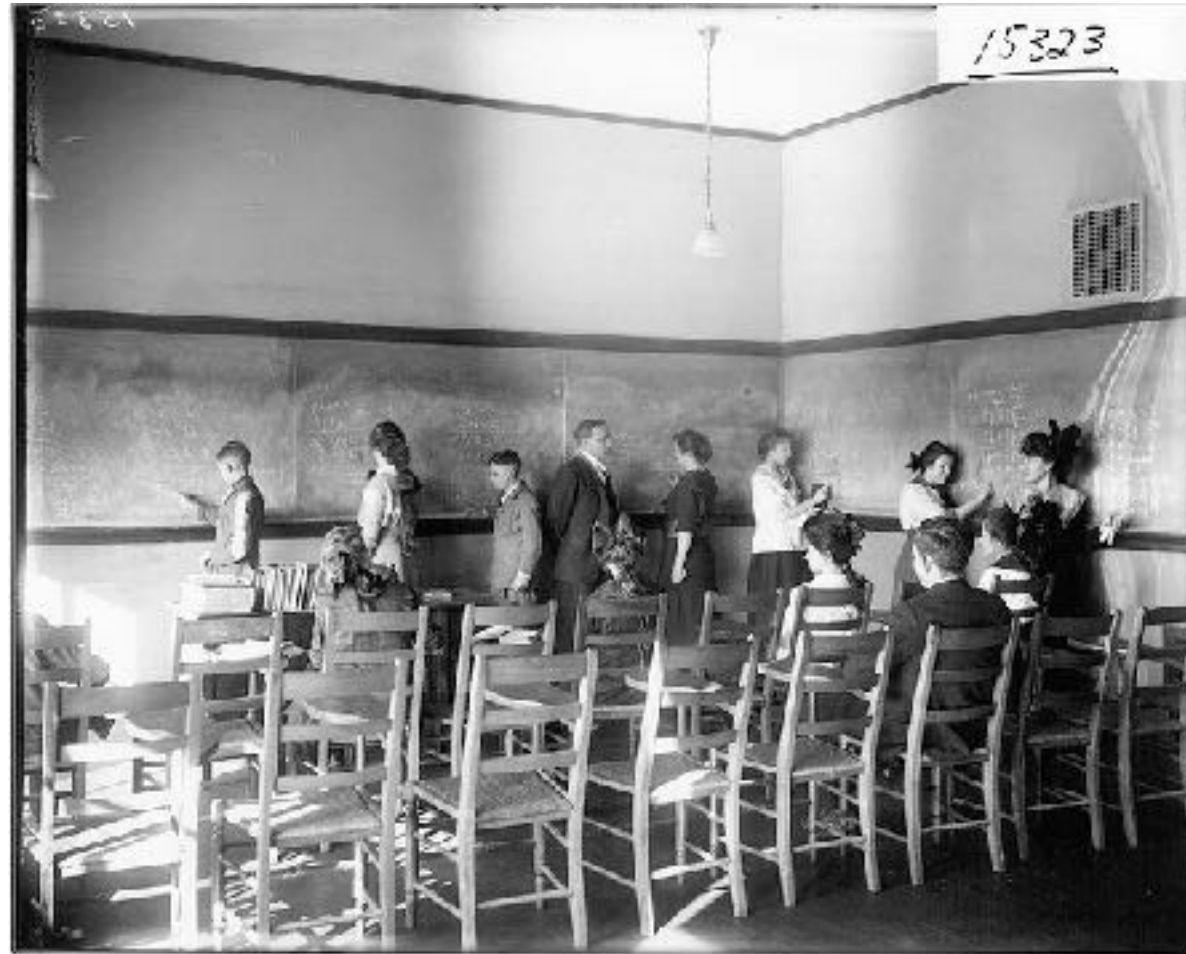


# Consultations



**Julian García**

**Tuesday:** 10AM to 11AM

Office 230, 25 Exhibition Walk, Clayton

# Lecture 6

# Arrays in MIPS

FIT 1008  
Introduction to Computer Science



COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

WARNING

This material has been reproduced and communicated to you by or on behalf of Monash University pursuant to Part VB of the Copyright Act 1968 (the Act). The material in this communication may be subject to copyright under the Act. Any further reproduction or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice.

# Objectives for this lecture

- How to write MIPS programs that involve **arrays**
- The need for **memory diagrams** and how to draw them
- Understand what are **pointers** and how do we use them.
- How to use **addressing modes** to access variables
- How to allocate memory on the **Heap**

# Lists vs **Arrays**

# Simple Implementation

- Arrays have a **fixed length**.
- All the items will have the **same size**.
- All the items in the list are initialised to 0.
- We will **store length** before the items.

# Arrays in MIPS

<code>len(a)</code>	5	<code>0x10012FBC</code>
<code>a[0]</code>	0	<code>0x10012FC0</code>
<code>a[1]</code>	-1	<code>0x10012FC4</code>
<code>a[2]</code>	4	<code>0x10012FC8</code>
<code>a[3]</code>	-9	<code>0x10012FCC</code>
<code>a[4]</code>	16	<code>0x10012FD0</code>

The high-level programmer's view: array is accessed through indices 0, 1, 2, ...

# Arrays in MIPS

<code>len(a)</code>	5	<code>0x10012FBC</code>
<code>a[0]</code>	0	<code>0x10012FC0</code>
<code>a[1]</code>	-1	<code>0x10012FC4</code>
<code>a[2]</code>	4	<code>0x10012FC8</code>
<code>a[3]</code>	-9	<code>0x10012FCC</code>
<code>a[4]</code>	16	<code>0x10012FD0</code>

The computer's view: array is part of memory, accessed through addresses  
`0x10012FC0`, `0x10012FC4`, ...

# Arrays in MIPS

**len(a)**

**a[0]**

**a[1]**

**a[2]**

**a[3]**

**a[4]**

**a**

5
0
-1
4
-9
16

**0x10012FBC**

**0x10012FBC**

**0x10012FC0**

**0x10012FC4**

**0x10012FC8**

**0x10012FCC**

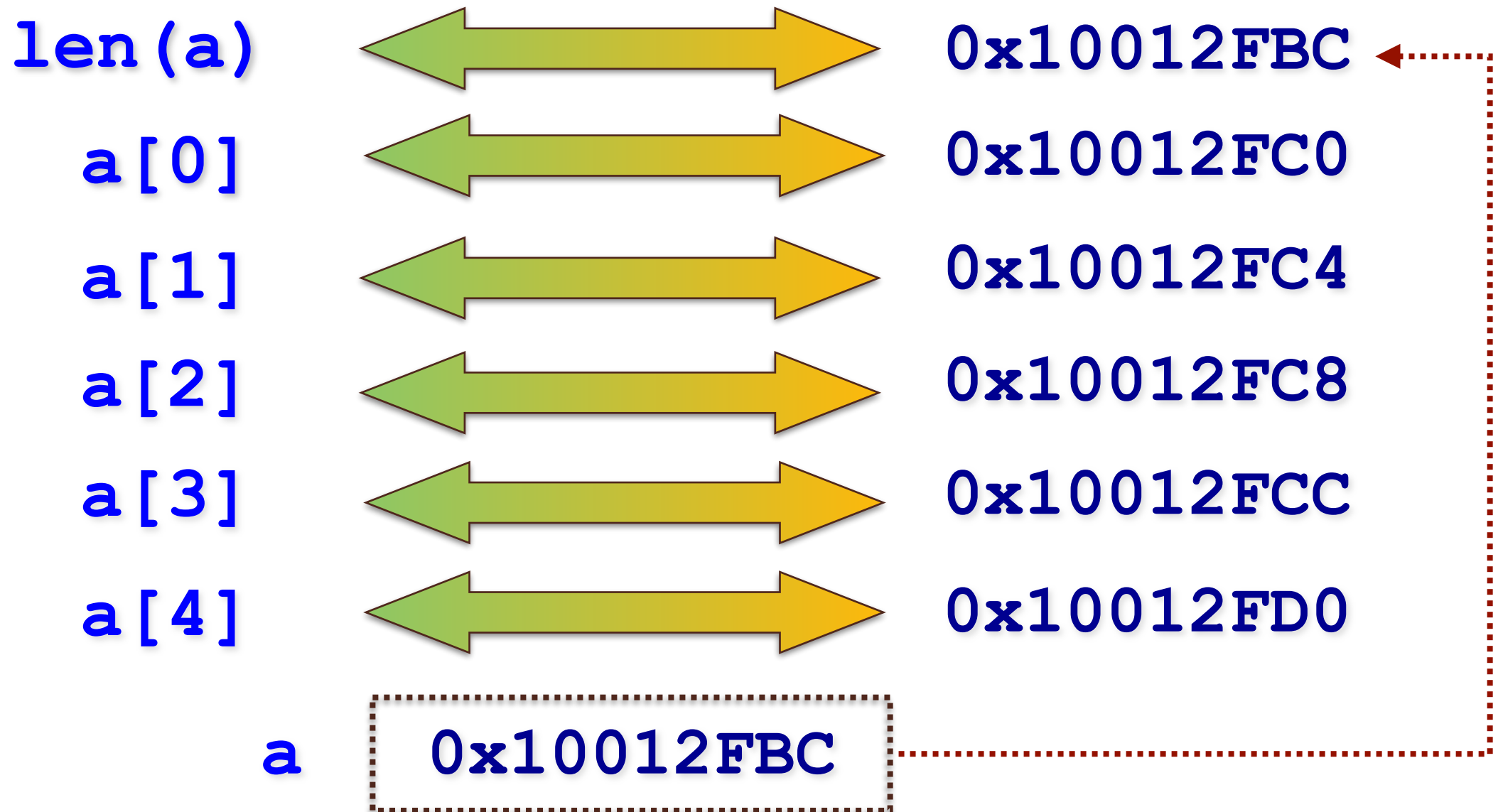
**0x10012FD0**

Variables  
which hold  
addresses  
are called  
pointers











# Arrays in MIPS



To program arrays in assembly language, need to understand their relationship, and how to convert from one to another.

# Arrays in MIPS

<code>len(a)</code>		<code>0x10012FBC</code>
<code>a[0]</code>		<code>0x10012FC0</code>
<code>a[1]</code>		<code>0x10012FC4</code>
<code>a[2]</code>		<code>0x10012FC8</code>
<code>a[3]</code>		<code>0x10012FCC</code>
<code>a[4]</code>		<code>0x10012FD0</code>

arrays: first  
index is  
always 0

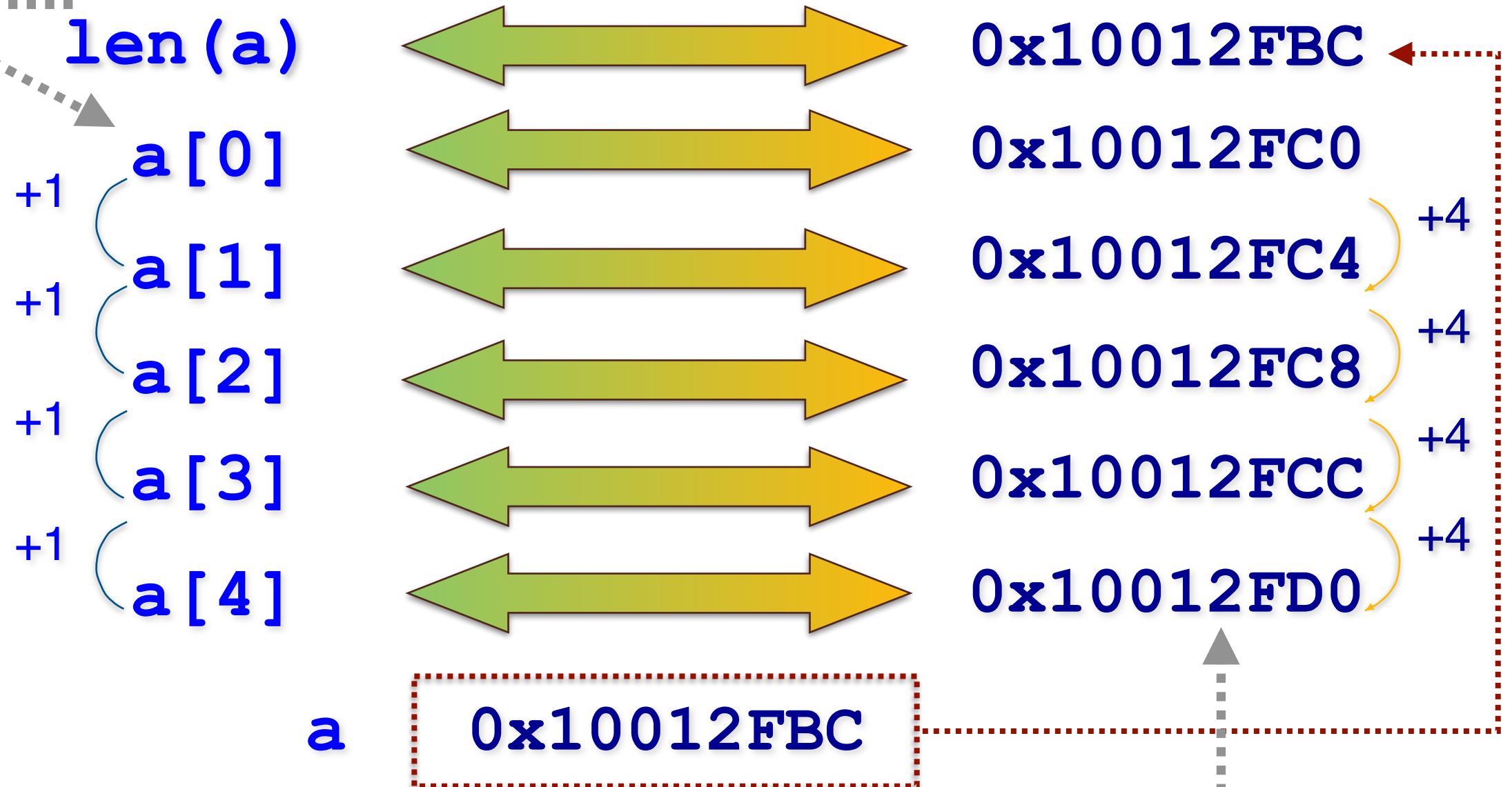
`a`

`0x10012FBC`

addresses: for a given array, address of first element is  
constant: address of array + 4 bytes to skip the **int** length

# Arrays in MIPS

arrays:  
adjacent  
indices  
differ by 1



addresses: addresses differ by size of array element type  
(here, 4 bytes for **integers**)

# Five ways to specify an address

- Directly (or using a label), e.g.  
`lw $t1, N` # loads from label N
- Label plus offset, e.g.  
`lw $t1, N+4` # loads from (label N + 4)
- Using a GPR to store the address, e.g.  
`lw $t1, ($s0)` # loads from address stored in \$s0
- GPR + offset, e.g.  
`lw $t1, 4($s0)`  
# loads from (address stored in \$s0)+4
- Label, offset, and GPR, e.g.  
`lw $t1, N+4($s0)`  
# loads from (label N+4)+contents of \$s0

# Creating Arrays in MIPS

```
the_list = [0]*size
```

- Allocate memory on the **Heap** for the list together with the length of list
- For integers, space required: **4\*size + 4**
- Store the address of the first byte of memory allocated in **the\_list**

# Arrays in MIPS

`len(the_list)`

5

`0x10012FBC`

`the_list[0]`

0

`0x10012FC0`

`the_list[1]`

-1

`0x10012FC4`

`the_list[2]`

4

`0x10012FC8`

`the_list[3]`

-9

`0x10012FCC`

`the_list[4]`

16

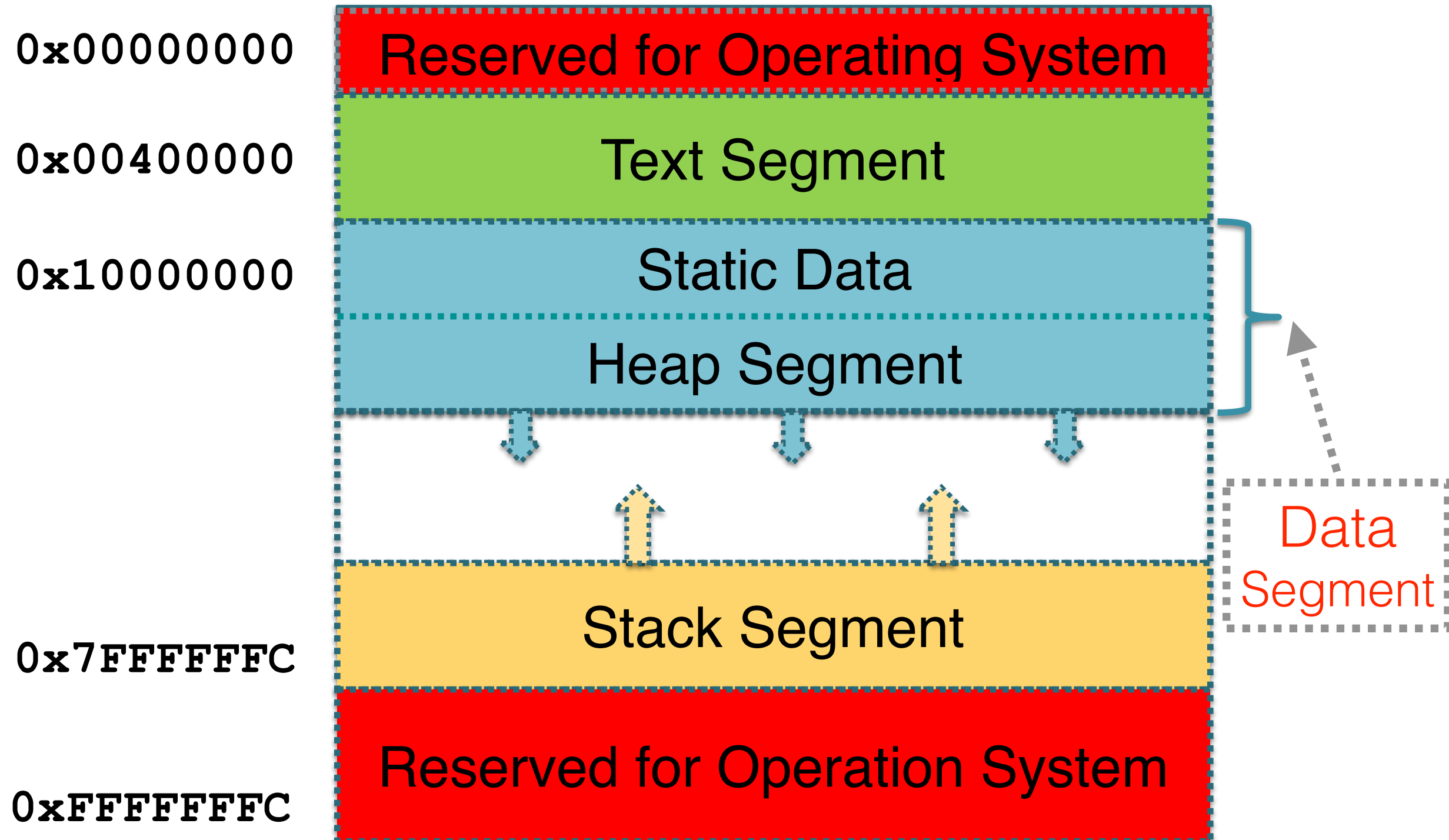
`0x10012FD0`

`the_list`

`0x10012FBC`



# MIPS Architecture: Memory



# construct\_list.py

```
size = int(input("Enter number of values: "))
the_list = [0] * size
for i in range(size):
    the_list[i] = int(input("Value: "))
print(the_list)
```



# Creating the list

```
lw    $t0, size
addi  $t1, $0, 4
mult   $t1, $t0
mflo   $t2
```

```
add $a0, $t2, $t1    # $a0 = 4*size + 4
addi $v0, $0, 9      # $v0 = 9
syscall              # allocate memory
```

```
sw    $v0, the_list # the_list now points to the returned address
sw    $t0, ($v0)     # store length of list
```

$\$t0 = \text{size}$   
 $\$t1 = 4$   
 $\$t2 = 4 * \text{size}$

allocate  $4 * \text{size} + 4$  bytes  
 (result in  $\$v0$ )

label `the_list`  
 references the memory  
 that will store the size

5	0x10012FBC
0	0x10012FC0
-1	0x10012FC4
4	0x10012FC8
-9	0x10012FCC
16	0x10012FD0
0x10012FBC	

Call code (\$v0)	Service	Arguments	Returns	Notes
9	Allocate memory	\$a0 = number of bytes	\$v0 = address of first byte	-

store the size in the  
 address referenced by  
 $\$t0$

# Lists in MIPS

`len(the_list)`

`the_list[0]`

`the_list[1]`

`the_list[2]`

`the_list[3]`

`the_list[4]`

`the_list`

5
0
-1
4
-9
16

0x10012FBC
------------

0x10012FBC

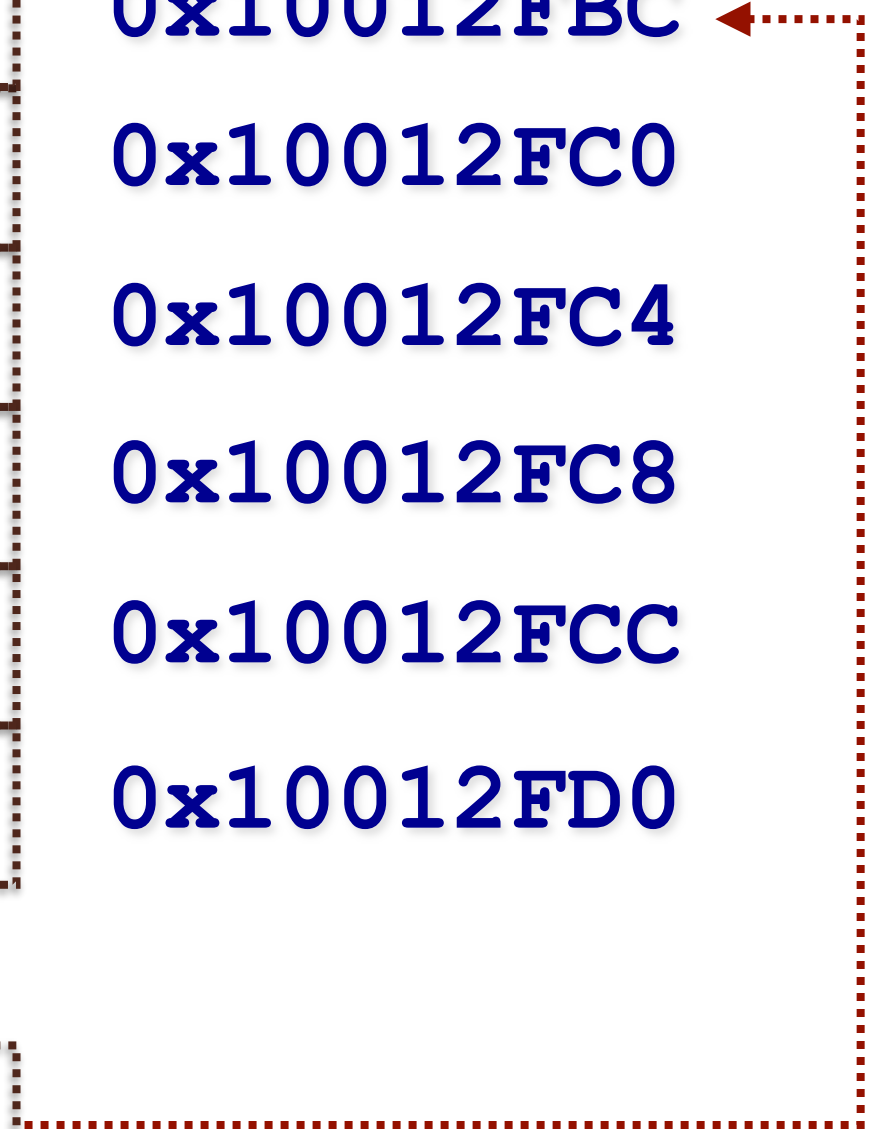
0x10012FC0

0x10012FC4

0x10012FC8

0x10012FCC

0x10012FD0



# Read in values into the list

```
for i in range(size):  
    the_list[i] = int(input("Value: "))
```

# Address of `the_list[k]`

`len(the_list)`

`the_list[0]`

`the_list[1]`

`the_list[2]`

`the_list[3]`

`the_list[4]`

`the_list`



$+4$

$k*4$

an integer  
size of 4  
bytes

address in `the_list`

+

4

+

$i*4$

# Read in values into the list

```
sw    $0, i          # i = 0
loop: lw    $t0, i
      lw    $t1, size
      # if i >= size goto endloop (Details omitted)
      # print prompt2 (Details omitted)
      # read next item into $v0 (Details omitted)
      lw    $t2, the_list
      addi  $t3, $0, 4
      mult  $t3, $t0
      mflo  $t4
      add   $t4, $t4, $t3 # t4 = i * 4 + 4
      add   $t4, $t4, $t2 # $t4 points to next location in the list
      sw    $v0, ($t4) # store the next value
      addi  $t0, $t0, 1 # i = i + 1
      sw    $t0, i
      j     loop
endloop:
```

# Print list

```
print(the_list)
```

# Print list

**lw:** This is correct, because *the\_list* is a pointer, its value is an address!

```
        addi $t0, $0, 0 # t0 = 0
loop2:   lw    $t1, the_list # $t1 = address of the_list
        lw    $t2, ($t1) # $t2 = size of list
        # if $t0 >= size goto endloop2 (Details omitted)
        addi $t3, $0, 4
        mult $t3, $t0
        mflo $t4
        add  $t4, $t4, $t3 # $t4 = $t0 * 4 + 4
        add  $t4, $t4, $t1
        lw   $a0, ($t4) # load current item value into $a0
        addi $v0, $0, 1
        syscall # print current item
        addi $a0, $0, 32 # print a space - ascii code 32
        addi $v0, $0, 11
        syscall
        addi $t0, $t0, 1 # $t0 = $t0 + 1
        j    loop2
endloop2:
```

(using directly \$t0 instead of i)

# Summary

- How we could represent lists in MIPS
- How to create lists
- How to access items in lists
- How to write MIPS programs involving lists