Assembly Language and Computer Architecture Lab

Nguyen Thi Thanh Nga
Dept. Computer engineering
School of Information and Communication Technology

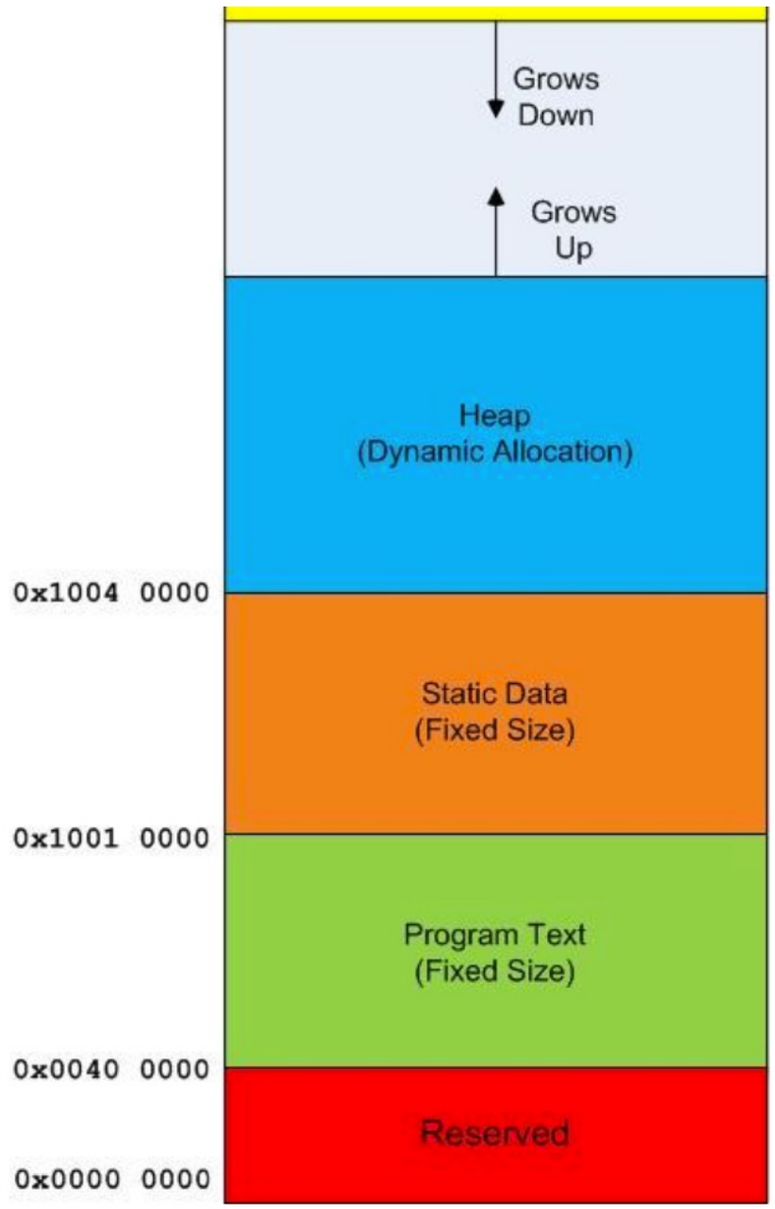
Week 10

- Heap memory and how to allocate and use it.
- The definition of an array, and how to implement and access elements in an array using assembly.
- How to allocate an array in stack memory, on the program stack, or in heap memory, and
- Why arrays are most commonly allocated on heap memory.
- How to use array addresses to access and print elements in an array.
- The Bubble Sort, and how to implement this sort in assembly.

Heap memory

- The heap memory segment begins in the program process space immediately after static memory.

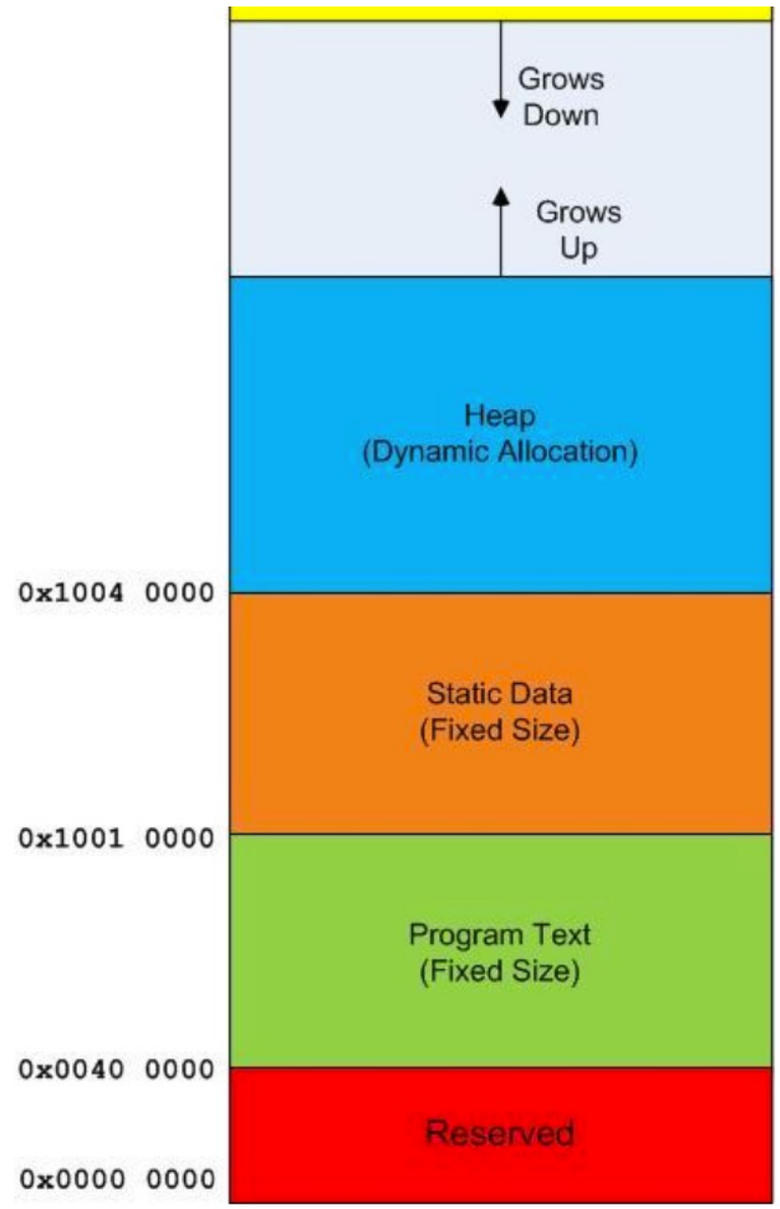
- It grows upward in memory until theoretically it reaches the stack memory segment.



Heap memory

- In reality most systems limit the amount of heap a process can request to protect against incorrect programs that might be allocating heap inappropriately.

- Generally, these limits on heap size can be increased if they need to be larger than the default.

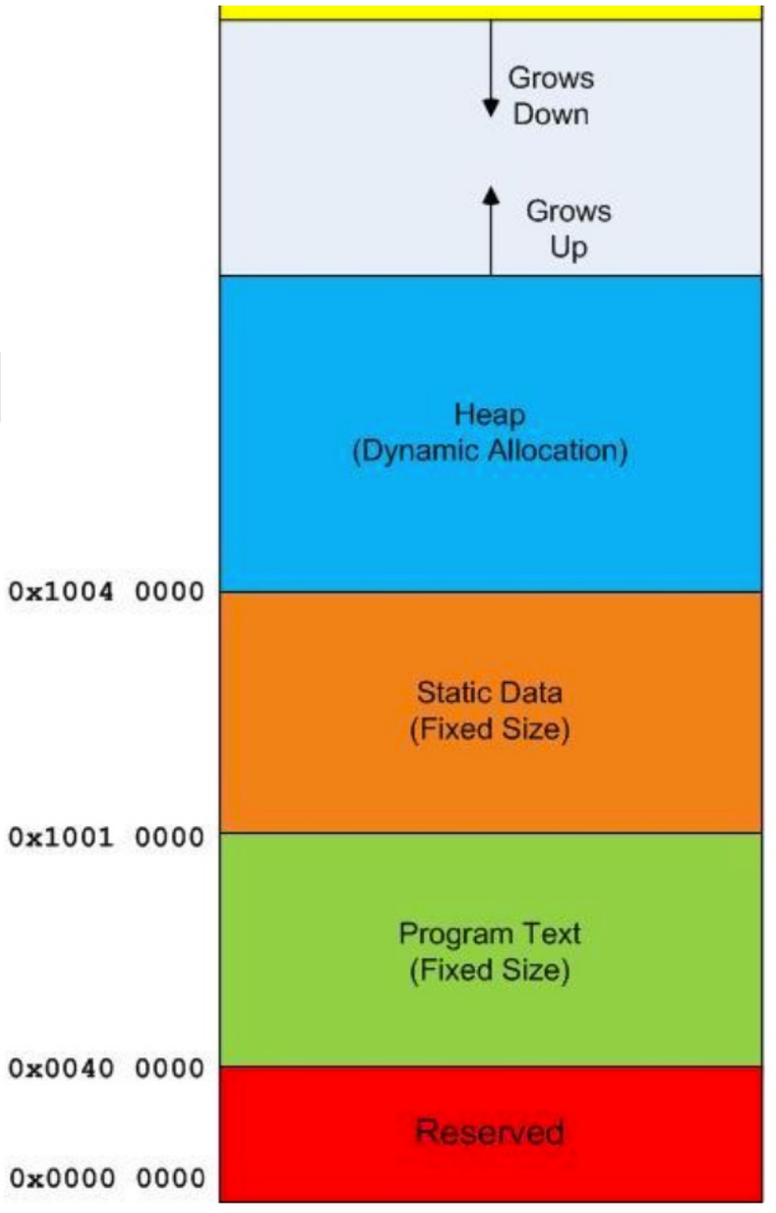


Heap memory

- Heap memory is dynamic, because it is allocated at run time based on a request from the programmer, normally using a *new* operator.

- The new operator allocates the requested amount of memory, and initializes it to a default value (normally zero).

- The amount of memory can be determined at run time.



Allocating heap memory

- The subprogram presented here using heap memory is a function that allows a programmer to prompt for strings without having to create a blank string variable in the **.data** section of the program.

- The subprogram first allocates a string variable large enough to hold the string the user is being prompted for.

- Then uses the syscall service 8 to read a value into that string.

Program 10.1
PromptString subprogram

```
.text
main:
    la $a0, prompt1      # Read and print the first string
    li $a1, 80
    jal PromptString
    move $a0, $v0
    jal PrintString

    la $a0, prompt2      # Read and print the second string
    li $a1, 80
    jal PromptString
    move $a0, $v0
    jal PrintString

    jal Exit

.data
    prompt1: .asciiz "Enter the first string: "
    prompt2: .asciiz "Enter the second string: "
```

Program 10.1
PromptString subprogram

```
.text
# Subprogram:     PromptString
# Purpose: To prompt for a string, allocate the string
# and return the string to the calling subprogram.
# Input:          $a0 - The prompt
#                 $a1 - The maximum size of the string
# Output:         $v0 - The address of the user entered string

PromptString:
     addi $sp, $sp, -12  # Push stack
     sw $ra, 0($sp)
     sw $a1, 4($sp)
     sw $s0, 8($sp)

     li $v0, 4           # Print the prompt in the function
     syscall             # $a0 still has the pointer
                         # to the prompt
```

Program 10.1
PromptString subprogram

```
li $v0, 9               # Allocate memory
lw $a0, 4($sp)
syscall
move $s0, $v0

move $a0, $v0           # Read the string
li $v0, 8
    lw $a1, 4($sp)
    syscall

    move $v0, $a0       # Save string address to return

    lw $ra, 0($sp)      # Pop stack
    lw $s0, 8($sp)
    addi $sp, $sp, 12
    jr $ra

.include "utils.asm"
```

Program 10.1
PromptString subprogram

```
.text
main:
    la $a0, prompt1     # Read and print the first string
    li $a1, 80
    jal PromptString
    move $a0, $v0
    jal PrintString

    la $a0, prompt2     # Read and print the second string
    li $a1, 80
    jal PromptString
    move $a0, $v0
    jal PrintString

    jal Exit

.data
    prompt1: .asciiz "Enter the first string: "
    prompt2: .asciiz "Enter the second string: "
```

- The main subprogram shows two strings being read.
- This is to show how the allocated strings exist in heap memory.
- The two strings entered are shown, with each taking up 80 bytes.

Program 10.1
PromptString subprogram

```
.text
# Subprogram:     PromptString
# Purpose: To prompt for a string, allocate the string
# and return the string to the calling subprogram.
# Input:          $a0 - The prompt
#                 $a1 - The maximum size of the string
# Output:         $v0 - The address of the user entered string

PromptString:
     addi $sp, $sp, -12
     sw $ra, 0($sp)
     sw $a1, 4($sp)
     sw $s0, 8($sp)

     li $v0, 4
     syscall
```

- Data which is expect to be unchanged across subprogram calls (including syscall) should always be stored in a save register (**$s0**), or on the stack (**$a1**).

- The value of **$s0** is saved when this subprogram is entered, and restored to its original value when the subprogram is left.
- All save registers must have the same value when leaving a subprogram as when it is entered.

Program 10.1
PromptString subprogram

```
li $v0, 9                  # Allocate memory
lw $a0, 4($sp)
syscall
move $s0, $v0
```

- The address of the memory returned from this heap allocation syscall is in **$v0**.

```
move $a0, $v0              # Read the string
li $v0, 8
lw $a1, 4($sp)
syscall
```

- **$v0** is moved to **$a0** to be used in the syscall service 8 to read a string.

```
move $v0, $a0              # Save string address to return

lw $ra, 0($sp)
lw $s0, 8($sp)
addi $sp, $sp, 12
jr $ra
```

- The address of the memory containing the string is now in **$a0**, and is moved to **$v0** to be returned to the main subprogram.

sbrk (allocate heap memory)	9	$a0 = number of bytes to allocate	$v0 contains address of allocated memory

Text Segment

Bkpt	Address	Code	Basic	Source
	0x004000a4	0x0000000c	syscall	38: syscall
	0x004000a8	0x03e00008	jr $31	40: jr $ra
	0x004000ac	0x24020004	addiu $2,$0,0x00000004	52: PromptInt:li $v0, 4
	0x004000b0	0x0000000c	syscall	53: syscall
	0x004000b4	0x24020005	addiu $2,$0,0x00000005	57: li $v0, 5
	0x004000b8	0x0000000c	syscall	58: syscall
	0x004000bc	0x03e00008	jr $31	60: jr $ra
	0x004000c0	0x20020004	addi $2,$0,0x00000004	69: PrintString:addi $v0, $zero, 4
	0x004000c4	0x0000000c	syscall	70: syscall
	0x004000c8	0x03e00008	jr $31	71: jr $ra
	0x004000cc	0x2402000a	addiu $2,$0,0x0000000a	79: Exit: li $v0, 10
	0x004000d0	0x0000000c	syscall	80: syscall

Labels

Label	Address ▲
mips10.1.asm	
main	0x00400000
PromptString	0x00400034
PrintNewLine	0x00400080
PrintInt	0x00400094
PromptInt	0x004000ac
PrintString	0x004000c0
Exit	0x004000cc
prompt1	0x10010000
prompt2	0x10010019
__PNL_newline	0x10010033

☑ Data ☑ Text

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10040000	e h T	s r i f	t s t	g n i r	\0 \0 \0 \n	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10040020	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10040040	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	e h T	o c e s	s d n	n i r t
0x10040060	\0 \0 \n g	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10040080	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100400a0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100400c0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100400e0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10040100	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10040120	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0

0x10040000 (heap) ☑ Hexadecimal Addresses ☑ Hexadecimal Values ☑ ASCII

Mars Messa

```
Enter the first string: The first string
The first string
Enter the second string: The second string
The second string

-- program is finished running --
```

Clear

Name	Number	Value
$zero	0	0x00000000
$at	1	0x10010000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x10040050
$a1	5	0x00000050

Arrays

- An array is a multivalued variable stored in a contiguous area of memory that contains elements that are all the same size.

- The minimum data needed to define an array consists of:

  - A variable which contains the address of the start of the array

  - The size of each element

  - The space to store the elements

Arrays

- To access any element in the array, the element address is calculated by the following formula:

```
elemAddress = basePtr + index * size
```

  - **elemAddress**: the address of (or pointer to) the element to be used.
  - **basePtr**: the address of the array variable
  - **index**: the index for the element (using 0 based arrays)
  - **size**: the size of each element

Arrays

- To load the element at index 0:

elemAddress=(0x10010044+(0*4))=0x10010044

→ basePtr for the array

- Đ To load the element at index 1:

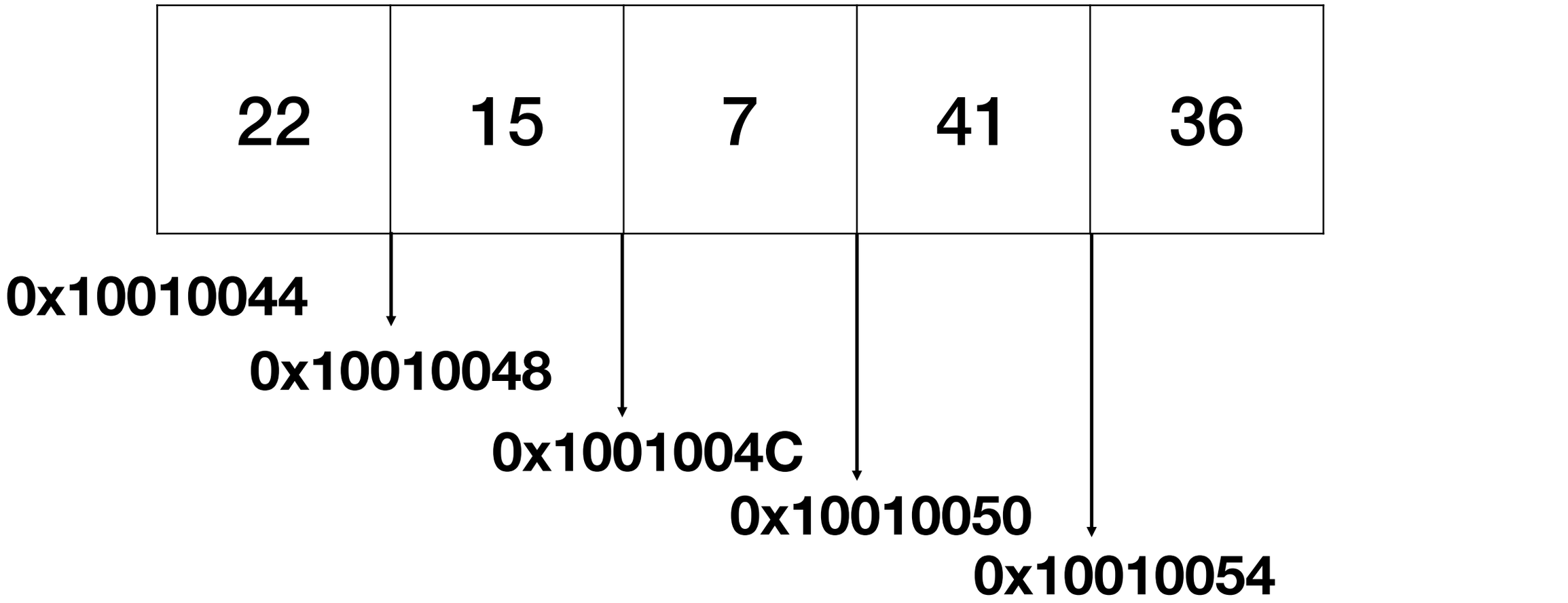
elemAddress=(0x10010044+(1*4))=0x10010048

- To load the element at index 2:

elemAddress=(0x10010044+(2*4))=0x1001004C

Arrays

- Mechanism of accessing array elements:




```
elemAddress = basePtr + index * size
```

Arrays

Consider the following example:

```
.data
.align  2               #Align next data item on specify
                        # 0=byte, 1=half, 2=word, 3=double
grades: .space 40
id:     .space 10
```

- The first array creates an array named ***grades***, which will store 10 elements, each 4 bytes big aligned on word boundaries.

- The second array creates an array named ***id*** of 10 bytes, no alignment is specified, so the bytes can cross word boundaries.

Arrays

- To access a grade element in the array grades:
  - grade 0: basePtr
  - grade 1: basePtr+4
  - grade 2: basePtr+8

- The following code fragment shows how grade 2 could be accessed:

```
addi $t0, 2          # set element number 2
sll $t0, $t0, 2      # multiply $t0 by 4 (size) to get the offset
la $t1, basePtr      # $t1 is the base of the array
add $t0, $t0, $t1    # basePtr + (index * size)
lw $t2, 0($t0)       # load element 2 into $t2
```

Allocating arrays in memory

- Arrays can be allocated in any part of memory.
- However, arrays allocated in the static data region or on the stack must be fixed size, with the size fixed at assembly time.
- Only heap allocated arrays can have their size set at run time.

Allocating arrays in static memory

- To allocate an array in static data, a label is defined to give the base address of the array, and enough space for the array elements is allocated.

- The array must take into account any alignment consideration.

- The following code fragment allocates an array of 10 integer words in the data segment.

```
.data
.align 2
array: .space 40
```

Allocating arrays stack

- To allocate an array on the stack, the **$sp** is adjusted so as to allow space on the stack for the array.

- In the case of the stack there is no equivalent to the **.align 2** assembler directive.

- The following code fragment allocates an array of 10 integer words on the stack after the **$ra** register.

```
addi $sp, $sp, -44
sw $ra, 0(sp)
# array begins at 4($sp)
```

Allocating arrays heap

- To allocate an array on the heap, the number of items to allocate is multiplied by the size of each element to obtain the amount of memory to allocate.

- A subprogram to do this, called AllocateArray, is shown below.

AllocatedArray subprogram

```
# Subprogram: AllocateArray

# Purpose: #To allocate an array of $a0 items,
#           each of size $a1
# Input:  $a0 - the number of items in the array
#         $a1 - the size of each item
# Output: $v0 - Address of the array allocated

AllocateArray:
addi $sp, $sp, -4
sw $ra, 0($sp)

mul $a0, $a0, $a1
li $v0, 9
syscall

lw $ra, 0($sp)
addi $sp, $sp, 4
jr $ra
```

Printing an Array

- This example shows how to access arrays by creating a PrintIntArray subprogram that prints the elements in an integer array.

- Two variables are passed into the subprogram, **$a0** which is the base address of the array, and **$a1**, which is the number of elements to print.

- The subprogram processes the array in a counter loop, and prints out each element followed by a ",".

Printing an Array

- The pseudo code for this subprogram follows.

```
Subprogram PrintIntArray(array, size)
{
    print("[")
    for (int i = 0; i < size; i++)
    {
        print("," + array[i])
    }
    print("]")
}
```

Printing an Array

```
.text
.globl main
main:
     la $a0, array_base
     lw $a1, array_size
     jal PrintIntArray
     jal Exit

.data
     array_size: .word 5
     array_base:
               .word 12
               .word 7
               .word 3
               .word 5
               .word 11
```

Printing an Array

```
.text
# Subprogram: PrintIntArray
# Purpose: print an array of ints
# inputs: $a0 - the base address of the array
#         $a1 - the size of the array
#
PrintIntArray:
    addi $sp, $sp, -16  # Stack record
    sw $ra, 0($sp)
    sw $s0, 4($sp)
    sw $s1, 8($sp)
    sw $s2, 12($sp)

    move $s0, $a0       # save the base of the array to $s0

    # initialization for counter loop
    # $s1 is the ending index of the loop
    # $s2 is the loop counter
    move $s1, $a1
    move $s2, $zero

    la $a0 open_bracket # print open bracket
    jal PrintString
```

Printing an Array

```
loop:
    # check ending condition
    sge $t0, $s2, $s1
    bnez $t0, end_loop

        sll $t0, $s2, 2 # Multiply the loop counter by
                        # by 4 to get offset (each element
                        # is 4 big)
        add $t0, $t0, $s0       # address of next array element
        lw $a1, 0($t0)  # Next array element
        la $a0, comma
        jal PrintInt    # print the integer from array

        addi $s2, $s2, 1        #increment $s0
        b loop
end_loop:
```

Printing an Array

```
    li $v0, 4               # print close bracket
    la $a0, close_bracket
    syscall

    lw $ra, 0($sp)
    lw $s0, 4($sp)
    lw $s1, 8($sp)
    lw $s2, 12($sp)                 # restore stack and return
    addi $sp, $sp, 16
    jr $ra
.data
    open_bracket: .asciiz "["
    close_bracket: .asciiz "]"
    comma: .asciiz ","
.include "utils.asm"
```

Mars Messages

```
[,12,7,3,5,11]
-- program is finished running --
```

Clear

Bubble Sort

- Sorting is the process of arranging data in an ascending or descending order.
- The following example will introduce the Bubble Sort, for sorting integer data in an array.

Bubble Sort

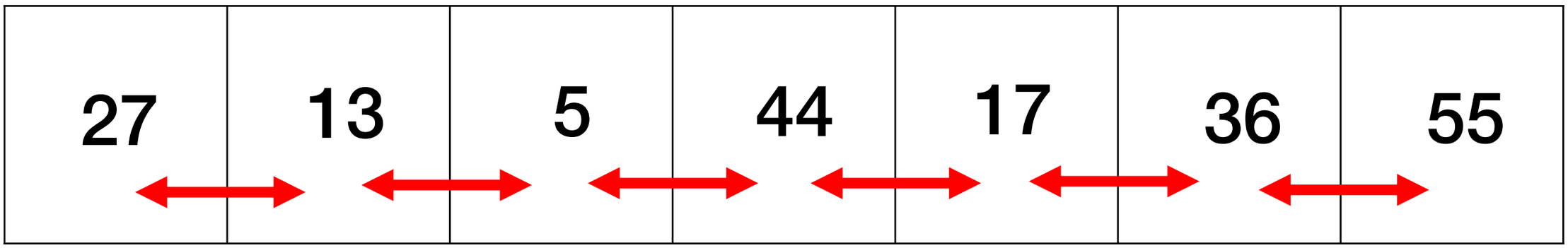
- Consider the following array containing integer values.

55	27	13	5	44	17	36

Bubble Sort

The sort is carried out in two loops.

- The inner loop passes once through the data comparing elements in the array and swapping them if they are not in the correct order.



Bubble Sort

- This process continues until a complete pass has been made through the array. At the end of the inner loop the largest value of the array is at the end of the array, and in its correct position. The array would look as follows.

27	13	5	44	17	36	55

Bubble Sort

The sort is carried out in two loops.

- An outer loop now runs which repeats the inner loop, and the second largest value moves to the correct position.

27	13	5	17	36	44	55

- Repeating this outer loop for all elements results in the array being sorted in ascending order.

Bubble Sort

- Pseudo code:

```
for (int i = 0; i < size-1; i++)
{
    for (int j = 0; j < ((size-1)-i); j++)
    {
        if (data[j] > data[j+1])
        {
            swap(data, j, j+1)
        }
    }
}

swap(data, i, j)
    int tmp = data[i];
    data[i] = data[j];
    data[j] = tmp;
}
```

Bubble Sort in MIPS assembly

```
.text
.globl main
main:
    la $a0, array_base
    lw $a1, array_size
    jal PrintIntArray

    la $a0, array_base
    lw $a1, array_size
    jal BubbleSort

    jal PrintNewLine
    la $a0, array_base
    lw $a1, array_size
    jal PrintIntArray

    jal Exit
```

```
.data
array_size: .word 7
array_base:
    .word 55
    .word 27
    .word 13
    .word 5
    .word 44
    .word 17
    .word 36
```

Bubble Sort in MIPS assembly

```
# Purpose:      Sort data using a Bubble Sort algorithm
# Input Params: $a0 - array
#               $a1 - array size
# Register conventions:
#           $s0 - array base
#           $s1 - array size
#           $s2 - outer loop counter
#           $s3 - inner loop counter
BubbleSort:
    addi $sp, $sp -20   # save stack information
    sw $ra, 0($sp)
    sw $s0, 4($sp)      # need to keep and restore save registers
    sw $s1, 8($sp)
    sw $s2, 12($sp)
    sw $s3, 16($sp)

    move $s0, $a0
    move $s1, $a1

    addi $s2, $zero, 0  #outer loop counter
```

Bubble Sort in MIPS assembly

```
OuterLoop:
    addi $t1, $s1, -1
    slt $t0, $s2, $t1
    beqz $t0, EndOuterLoop

    addi $s3, $zero, 0 #inner loop counter
    InnerLoop:
        addi $t1, $s1, -1
        sub $t1, $t1, $s2
        slt $t0, $s3, $t1
        beqz $t0, EndInnerLoop

        sll $t4, $s3, 2 # load data[j]. Note offset is 4 bytes
        add $t5, $s0, $t4
        lw $t2, 0($t5)

        addi $t6, $t5, 4  # load data[j+1]
        lw $t3, 0($t6)

        sgt $t0, $t2, $t3
        beqz $t0, NotGreater
            move $a0, $s0
            move $a1, $s3
            addi $t0, $s3, 1
            move $a2, $t0
            jal Swap          # t5 is &data[j], t6 is &data[j=1]
        NotGreater:
        addi $s3, $s3, 1
        b InnerLoop
    EndInnerLoop:
    addi $s2, $s2, 1
    b OuterLoop
EndOuterLoop:
```

```
lw $ra, 0($sp)      #restore stack information
lw $s0, 4($sp)
lw $s1, 8($sp)
lw $s2, 12($sp)
lw $s3, 16($sp)
addi $sp, $sp 20
jr $ra
```

Bubble Sort in MIPS assembly

```
# Subprogram:    Swap
# Purpose:       To swap values in an array of integers
# Input parameters: $a0 - the array containing elements to swap
#                   $a1 - index of element 1
#                   $a2 - index of elelemnt 2
# Side Effects: Array is changed to swap element 1 and 2
Swap:
    sll $t0, $a1, 2      # calcualate address of element 1
add $t0, $a0, $t0
sll $t1, $a2, 2      # calculate address of element 2
add $t1, $a0, $t1

lw $t2, 0($t0)       #swap elements
lw $t3, 0($t1)
sw $t2, 0($t1)
sw $t3, 0($t0)

jr $ra
```

Bubble Sort in MIPS assembly

```
# Subprogram:    PrintIntArray
# Purpose:       Print an array of ints
# inputs: $a0 - the base address of the array
#         $a1 - the size of the array
#
PrintIntArray:
    addi $sp, $sp, -16  # Stack record
    sw $ra, 0($sp)
    sw $s0, 4($sp)
    sw $s1, 8($sp)
    sw $s2, 12($sp)

    move $s0, $a0       # save the base of the array to $s0


    # initialization for counter loop
    # $s1 is the ending index of the loop
    # $s2 is the loop counter
    move $s1, $a1
    move $s2, $zero

    la $a0 open_bracket # print open bracket
    jal PrintString
-
```

Bubble Sort in MIPS assembly

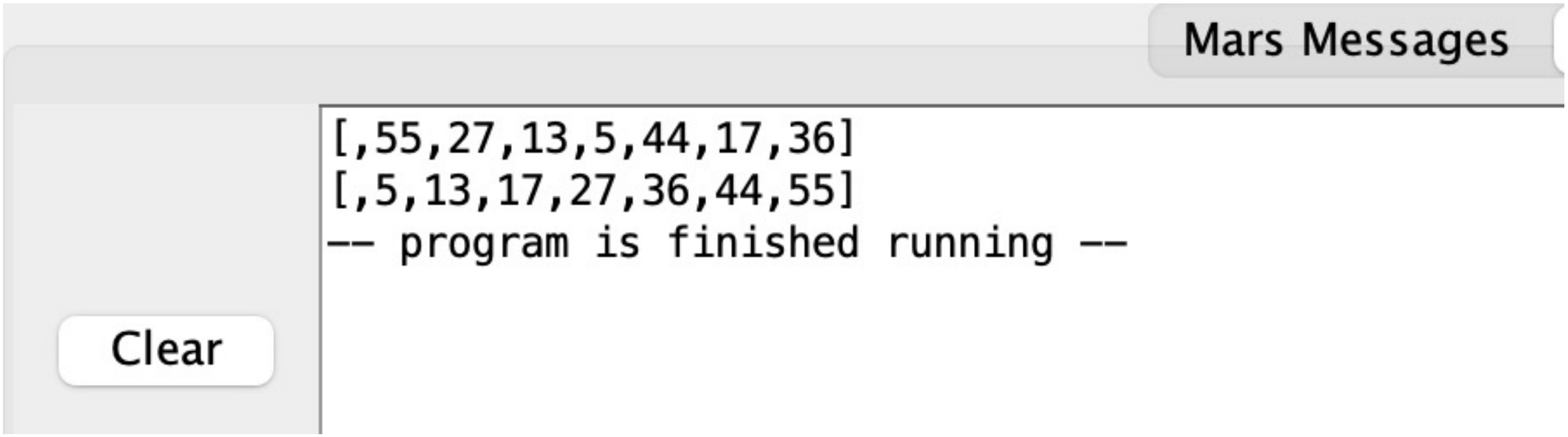
```
loop:
    # check ending condition
    sge $t0, $s2, $s1
    bnez $t0, end_loop

        sll $t0, $s2, 2          # Multiply the loop counter by
                                 # by 4 to get offset (each element
                                 # is 4 big)
        add $t0, $t0, $s0        # address of next array element
        lw $a1, 0($t0)           # Next array element
        la $a0, comma
        jal PrintInt              # print the integer from array

        addi $s2, $s2, 1         #increment $s0
        b loop
end_loop:
    li $v0, 4                    # print close bracket
    la $a0, close_bracket
    syscall

    lw $ra, 0($sp)
    lw $s0, 4($sp)
    lw $s1, 8($sp)
    lw $s2, 12($sp)
    addi $sp, $sp, 16            # restore stack and return
    jr $ra
.data
    open_bracket: .asciiz "["
    close_bracket: .asciiz "]"
     comma: .asciiz ","
.include "utils.asm"
```

Bubble Sort in MIPS assembly



Excercise 10.1

- Change the PrintIntArray subprogram so that it prints the array from the last element to the first element.

Excercise 10.2

- The following pseudo code converts an input value of a single decimal number from $1 \leq n \leq 15$ into a single hexadecimal digit.
- Translate this pseudo code into MIPS assembly.

Excercise 10.2

```
main
{
     String a[16]
     a[0] = "0x0"
     a[1] = "0x1"
     a[2] = "0x2"
     a[3] = "0x3"
     a[4] = "0x4"
     a[5] = "0x5"
     a[6] = "0x6"
     a[7] = "0x7"
     a[8] = "0x8"
     a[9] = "0x9"
     a[10] = "0xa"
     a[11] = "0xb"
     a[12] = "0xc"
     a[13] = "0xd"
     a[14] = "0xe"
     a[15] = "0xf"

     int i = prompt("Enter a number from 0 to 15 ")
     print("your number is " + a[i]
}
```

Exercise 10.3

- Implement the Buble Sort program that the user is prompted for the maximum size of the array, and then fill the array with random numbers.

- Sort the array using Buble sort.

- Print out the array.

Exercise 10.4

- The AllocateArray subprogram is incorrect in that the allocation can fall on any boundary. This is a problem if the array is of elements that must fall on a specific boundary. For example, if the array is contains ints, the array allocation must fall on full word boundary.

a) Using the PromptString and AllocateArray subprograms, show how this problem can occur.

b) Change the AllocateArray program to always do allocations on a double word boundary.

Assigment 10.1

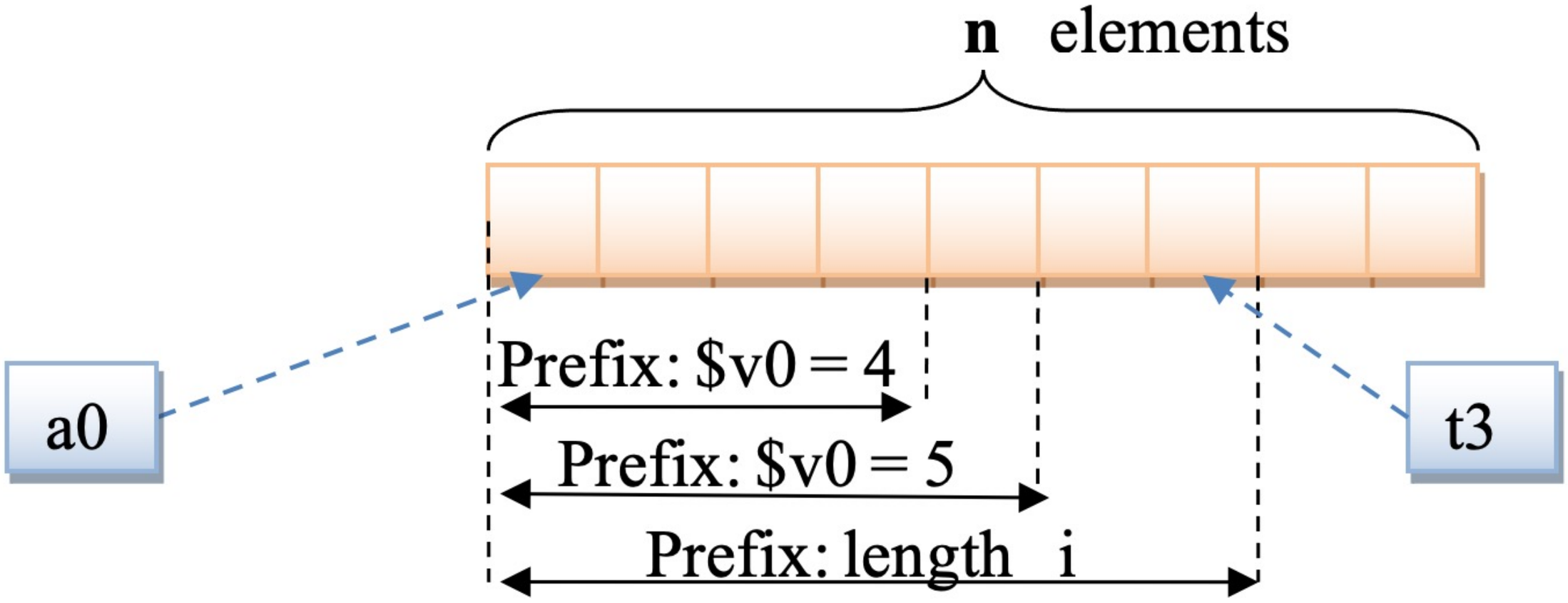
- Consider a list of integers of length n. A prefix of length I for the given list consist of the first i integers in the list, where $0 \leq i \leq n$.

- A maximum-sum prefix, as the name implies, is a prefix for which the sum of elements is the largest among all prefixes.

- For example, if the list is (2,-3,2,5,-4), its maximum-sum prefix consists of the first four elements and the associated sum is 2-3+2+5=6; no other prefix of the given list has a larger sum.

Assigment 10.1

- The following procedure uses indexing method to find the maximum-sum prefix in a list of integers. Read this procedure carefully, especially indexing method.



Assigment 10.1

- Create a new project to implement the Subprogram.
- Code the main program and initialize data for the integer list.
- Compile and upload to simulator.
- Run the program step by step, observe the process of explore each element of the integer list using indexing method.

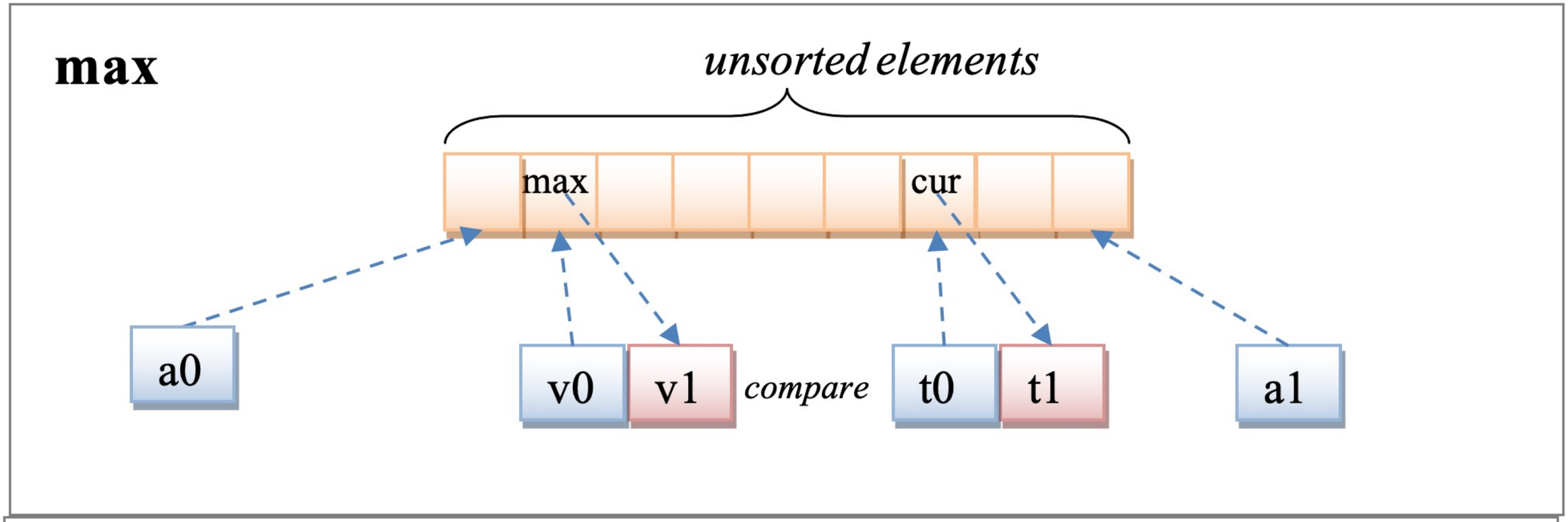
Assigment 10.2

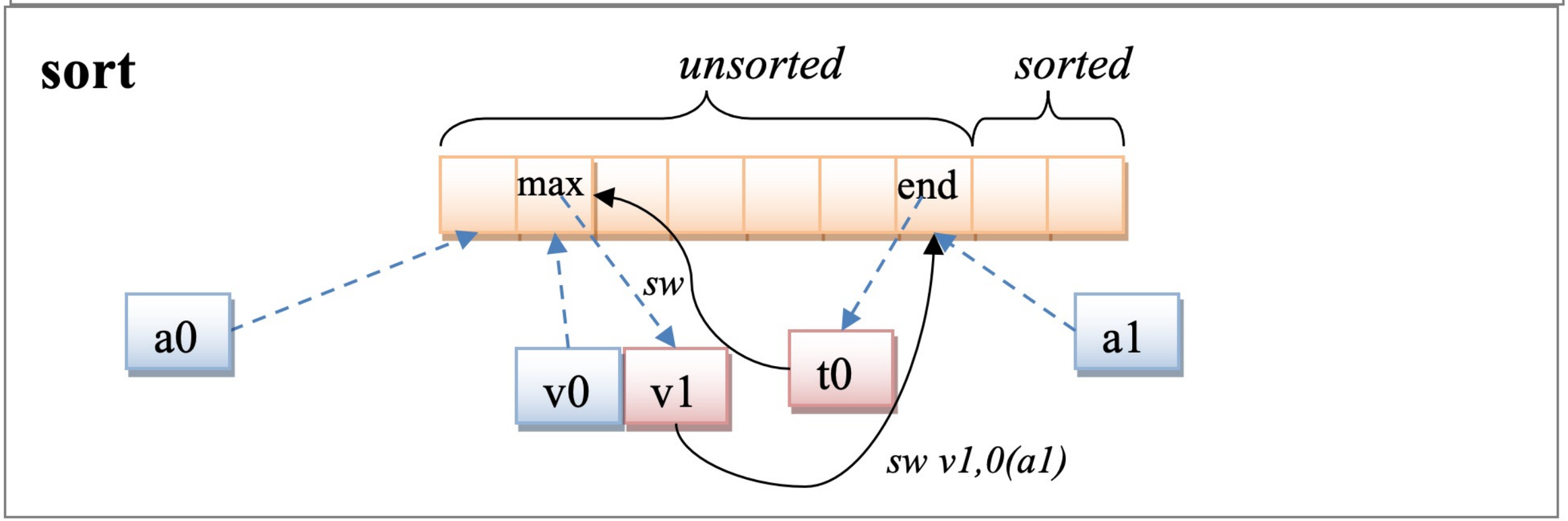
- A given list of n numbers can be sorted in ascending order as follows.

- Find a largest number in the list (there may be more than one) and swap it with the last element in the list. The new last element is now in its proper position in sorted order.

- Now, sort the remaining n-1 elements using the same step repeatedly. When only one element is left, sorting is complete. This method is known as selection sort.

Assigment 10.2





Assigment 10.2

- Create a new project to implement the procedure.
- Code the program and initialize data for the integer list.
- Compile and upload to simulator.
- Run the program step by step, observe the process of explore each element of the integer list using pointer updating method.

Assigment 10.3

- Write a program to find prime numbers from 3 to n in a loop by dividing the number n by all numbers from 2...n/2 in an inner loop.

  - Using the remainder (rem) operation, determine if n is divisible by any number.

  - If n is divisible, leave the inner loop.

  - If the limit of n/2 is reached and the inner loop has not been exited, the number is prime and you should output the number.

  - So if the user were to enter 25, your program would print out "2, 3, 5, 7, 11, 13, 17, 19, 23".

Assigment 10.4

- The following pseudo code programs calculates the Fibonacci numbers from 1..n, and stores them in an array. Translate this pseudo code into MIPS assembly, and use the PrintIntArray subprogram to print the results.

Assigment 10.4

```
main
{
    int size = PromptInt("Enter a max Fibonacci number to calc: ")
    int Fibonacci[size]
    Fibonacci[0] = 0
    Fibonacci[1] = 1
    for (int i = 2; i < size; i++)
    {
        Fibonacci[i] = Fibonacci[i-1] + Fibonacci[i-2]
}
PrintIntArray(Fibonacci, size)
```

Assigment 10.5

- Implement a Binary Search algorithm, and using the results from Exercise 10.3, show how long the Binary Search takes (on average) for arrays of size 10, 100, and 1000. (You do not have to print out the values in the array).

End of week 10