

Thực hành Kiến trúc máy tính

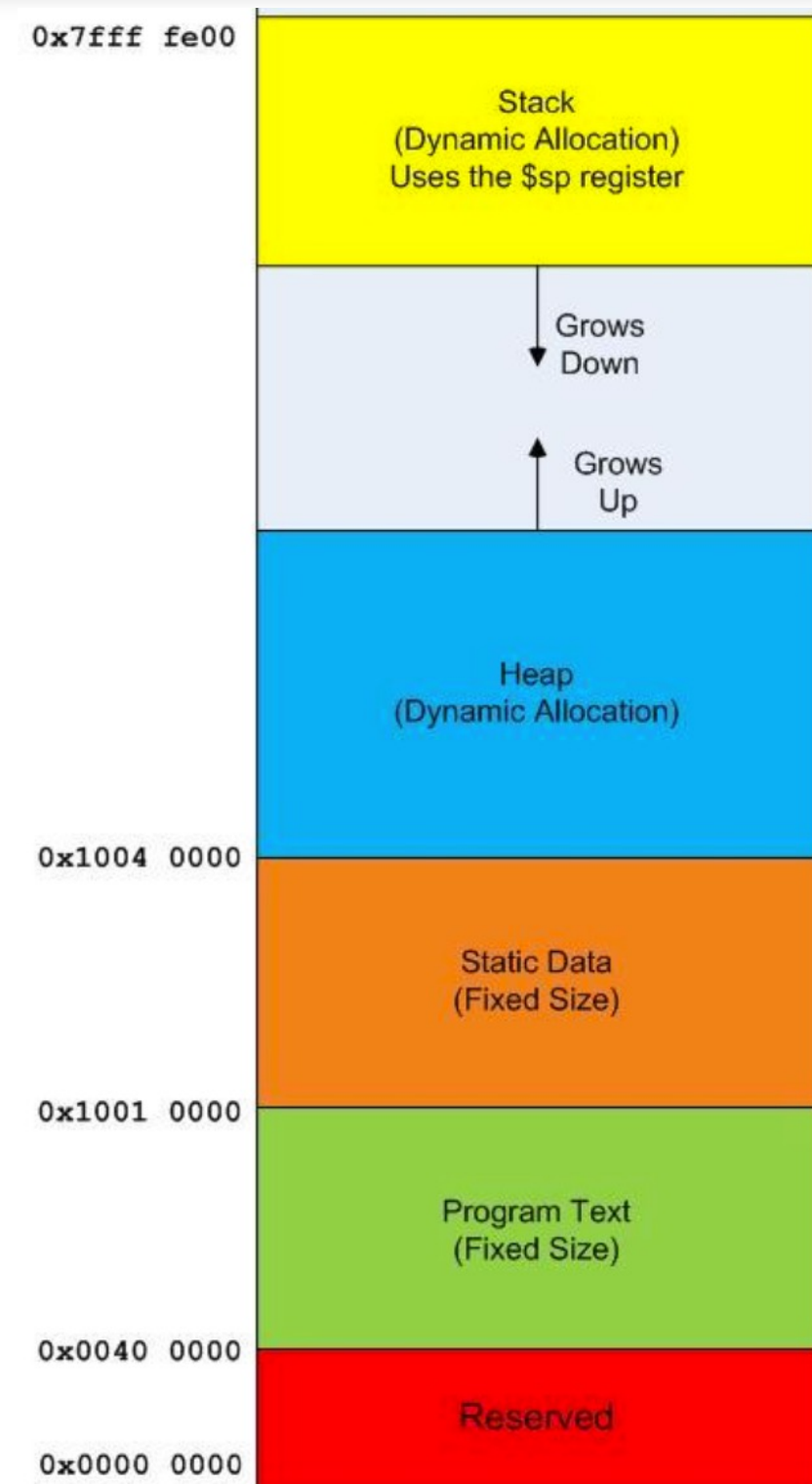
Giảng viên: Nguyễn Thị Thanh Nga
Khoa Kỹ thuật máy tính
Trường CNTT&TT

Tuần 10

- Bộ nhớ heap và cách cấp phát và sử dụng nó.
- Định nghĩa về mảng, cách triển khai và truy cập các phần tử trong một mảng sử dụng hợp ngữ.
- Cách cấp phát một mảng trong bộ nhớ ngăn xếp, trên ngăn xếp chương trình hoặc trong bộ nhớ heap
- Tại sao mảng được cấp phát phổ biến nhất trên bộ nhớ heap.
- Cách sử dụng địa chỉ mảng để truy cập và in các phần tử trong mảng.
- Sắp xếp bong bóng và cách triển khai thuật toán này trong hợp ngữ

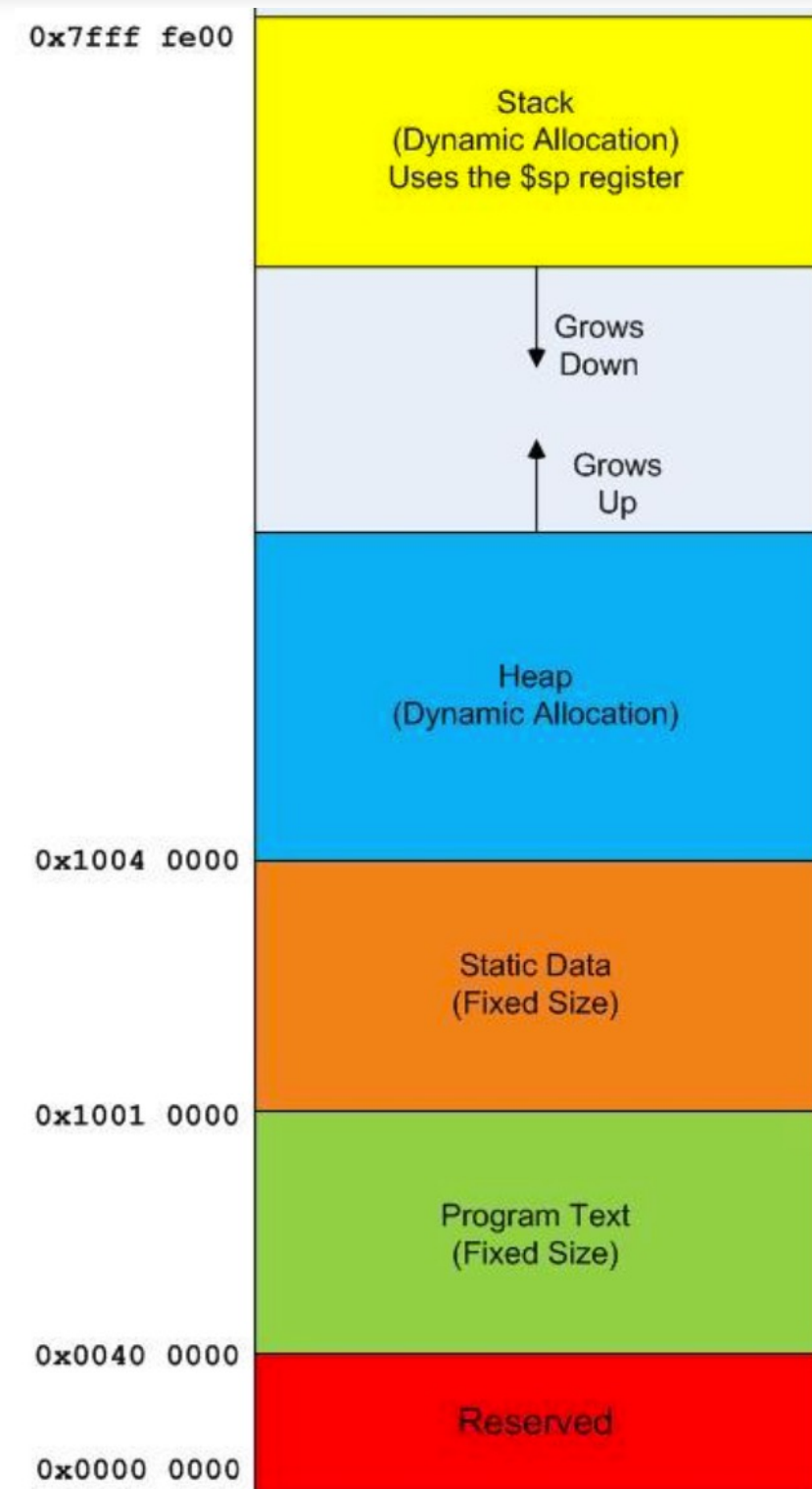
Bộ nhớ Heap

- Bộ nhớ heap bắt đầu ngay sau bộ nhớ tĩnh trong không gian bộ nhớ.
- Về mặt lý thuyết, bộ nhớ heap tăng dần cho đến khi gặp bộ nhớ ngăn xếp.



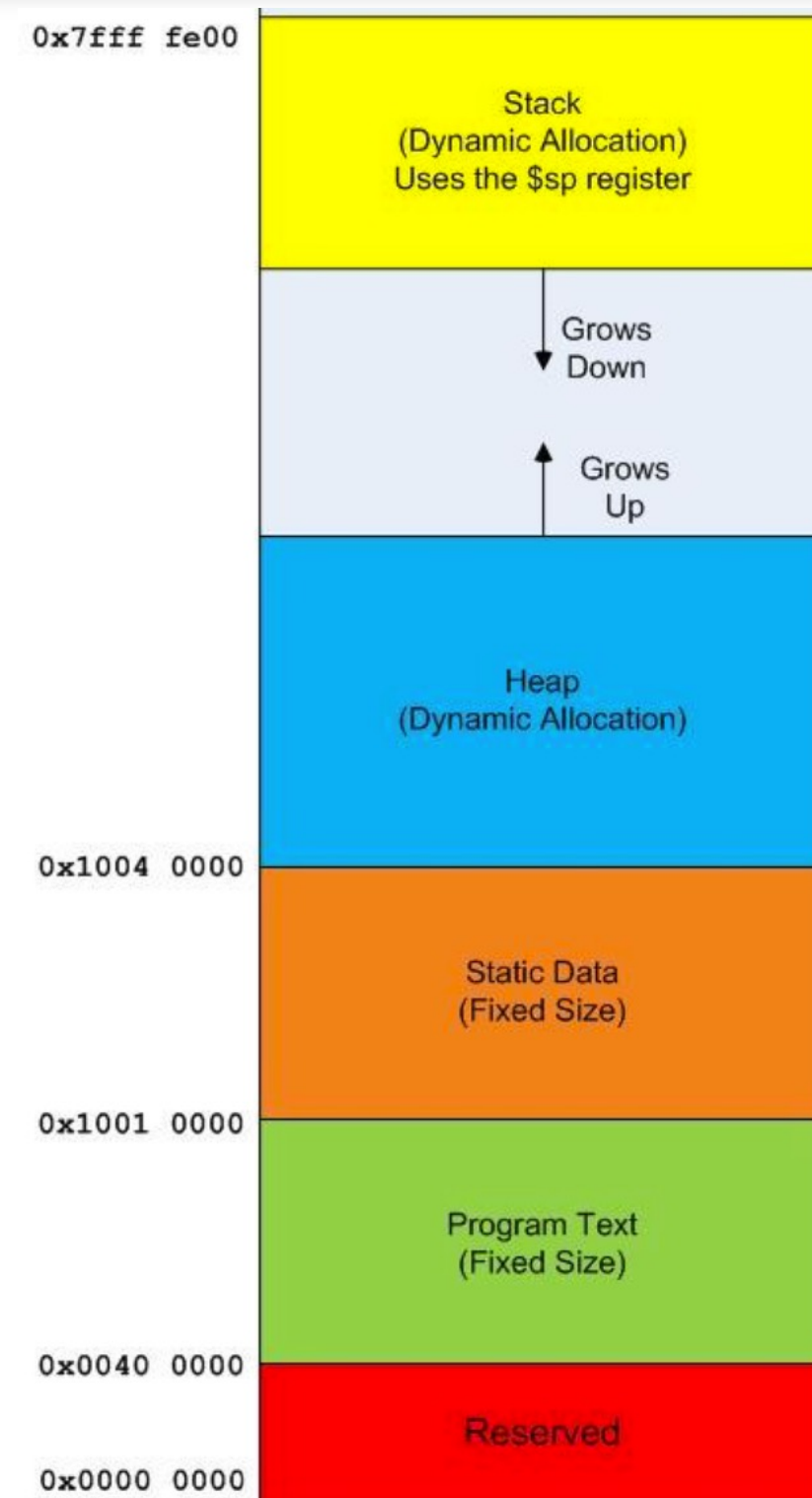
Bộ nhớ Heap

- Trong thực tế, hầu hết các hệ thống đều giới hạn không gian nhớ heap mà một chương trình có thể yêu cầu để bảo vệ khỏi các chương trình không chính xác có thể phân bổ heap không thích hợp.
- Nói chung, các giới hạn về kích thước bộ nhớ heap có thể tăng nếu cần không gian nhớ lớn hơn mặc định.



Bộ nhớ Heap

- Bộ nhớ heap là bộ nhớ động, vì nó được cấp phát tại thời điểm thực thi dựa trên yêu cầu từ người lập trình, thường là do sử dụng toán tử mới.
- Toán tử mới cấp phát bộ nhớ yêu cầu và khởi tạo nó theo một giá trị mặc định (thường là 0).
- Không gian nhớ có thể được xác định tại thời điểm thực thi.



Cấp phát bộ nhớ heap

- Chương trình con sau đây sử dụng bộ nhớ heap là một hàm cho phép người lập trình nhắc người dùng nhập vào một chuỗi mà không cần phải tạo biến chuỗi rỗng trong chỉ thị **.data** của chương trình.
- Chương trình con trước tiên cấp phát một biến chuỗi đủ lớn để chứa chuỗi mà người dùng đang được nhắc.
- Sau đó, sử dụng dịch vụ syscall 8 để đọc giá trị vào chuỗi đó.

Program 10.1

Chương trình con PromptString

.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

Chương trình con PromptString

.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

Chương trình con PromptString

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

Allocate memory

```
move $a0, $v0
li $v0, 8
lw $a1, 4($sp)
syscall
```

Read the string

read string	8	\$a0 = address of input buffer \$a1 = maximum number of characters to read
-------------	---	---

```
move $v0, $a0
```

Save string address to return

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

Pop stack

```
.include "utils.asm"
```

Program 10.1

Chương trình con PromptString

.text
main:

```
la $a0, prompt1
li $a1, 80
jal PromptString
move $a0, $v0
jal PrintString
```

```
la $a0, prompt2
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: "
```

- Chương trình chính hiển thị hai chuỗi được đọc.

Read and print the first string

- Hiển thị cách các chuỗi được cấp phát trong bộ nhớ heap.

Read and print the second string

- Hiển thị hai chuỗi đã nhập vào, mỗi chuỗi 80 byte.

Program 10.1

Chương trình con PromptString

```
.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
```

- Dữ liệu không thay đổi trong lời gọi chương trình con (bao gồm cả syscall) phải luôn được lưu trữ trong thanh ghi lưu (\$s0) hoặc trên ngăn xếp (\$a1).

- Giá trị của \$s0 được lưu lại khi vào chương trình con và được khôi phục về giá trị ban đầu khi kết thúc chương trình con.
- Tất cả các thanh ghi lưu phải có cùng giá trị trước khi vào 1 chương trình con và sau khi kết thúc chương trình con đó.

Program 10.1

Chương trình con PromptString

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

Allocate memory

- Địa chỉ của bộ nhớ được trả về từ syscall 9, cấp phát bộ nhớ heap, được lưu trong \$v0.

```
move $a0, $v0
li $v0, 8
lw $a1, 4($sp)
syscall
```

Read the string

- \$v0 được chuyển đến \$a0 để gọi syscall 8 đọc vào một chuỗi ký tự.

```
move $v0, $a0

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

Save string address to return

- Địa chỉ của vùng nhớ chứa chuỗi lưu trong \$a0 được chuyển sang \$v0, trở thành giá trị trả về của chương trình con.

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

Text Segment

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

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

Mảng

- Mảng là một biến nhiều giá trị được lưu trữ trong một vùng bộ nhớ liên tục có các phần tử có cùng kích thước.
- Dữ liệu tối thiểu cần thiết để định nghĩa một mảng bao gồm:
 - Một biến chứa địa chỉ đầu mảng
 - Kích thước của mỗi phần tử
 - Không gian để lưu trữ các phần tử

Mảng

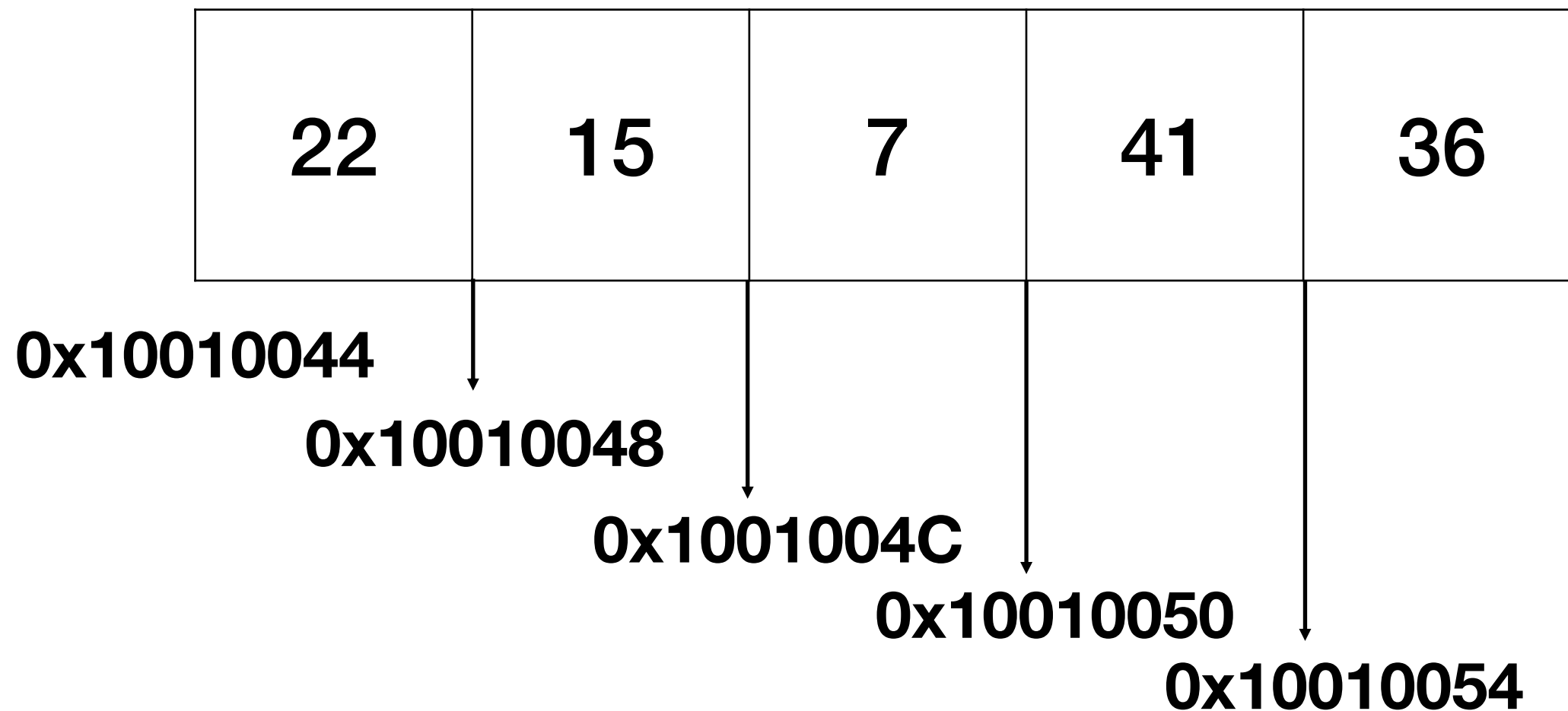
- Để truy cập bất kỳ phần tử nào trong mảng, địa chỉ của phần tử được tính theo công thức sau:

$$\text{elemAddress} = \text{basePtr} + \text{index} * \text{size}$$

- **elemAddress**: địa chỉ của (hoặc con trỏ tới) phần tử sẽ được sử dụng.
- **basePtr**: địa chỉ của biến mảng
- **index**: chỉ số cho phần tử (sử dụng mảng dựa trên 0)
- **kích thước**: kích thước của mỗi phần tử

Mảng

- Cơ chế truy cập các phần tử mảng:



$$\text{elemAddress} = \text{basePtr} + \text{index} * \text{size}$$

Mảng

- Để tải phần tử ở index 0:

$$\text{elemAddress} = (0x10010044 + (0 * 4)) = 0x10010044$$

→ basePtr cho mảng

- Để tải phần tử ở index 1:

$$\text{elemAddress} = (0x10010044 + (1 * 4)) = 0x10010048$$

- Để tải phần tử ở index 2:

$$\text{elemAddress} = (0x10010044 + (2 * 4)) = 0x1001004C$$

Mảng

Xét ví dụ sau:

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

- Khai báo đầu tiên tạo một mảng có tên là **grades**, có kích thước 40 byte. Mảng này sẽ lưu trữ 10 phần tử, mỗi phần tử có kích thước 4 byte, căn chỉnh theo từ (.align 2 => word boundary).
- Khai báo thứ hai tạo ra một mảng tên là **id**, kích thước 10 byte, không có căn chỉnh nào được chỉ định, vì vậy các byte có thể vượt qua ranh giới từ.

Mảng

- Để truy cập một phần tử trong mảng grades:
 - Phần tử thứ 0: `basePtr`
 - Phần tử thứ 1: `basePtr + 4`
 - Phần tử thứ 2: `basePtr + 8`
- Đoạn mã sau đây cho thấy cách phần tử thứ 2 có thể được truy cập:

```
addi $t0,$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
```

Cấp phát mảng trong bộ nhớ

- Mảng có thể được cấp phát trong bất kỳ phần nào của bộ nhớ.
- Tuy nhiên, các mảng được cấp phát trong vùng dữ liệu tĩnh hoặc trên ngăn xếp phải có kích thước cố định, và được cấp phát kích thước cố định ở thời điểm biên dịch.
- Chỉ các mảng được cấp phát trong bộ nhớ heap mới có thể được phân bổ kích thước tại thời điểm chạy chương trình.

Cấp phát mảng trong bộ nhớ tĩnh

- 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
```

Cấp phát mảng trên ngăn xếp

- Để cấp phát một mảng trên ngăn xếp, `$sp` được điều chỉnh để cấp phát một không gian trên ngăn xếp cho mảng.
- Đối với ngăn xếp, không có chỉ thị nào tương đương với chỉ thị trình hợp dịch `.align 2`.
- Đoạn mã sau cấp phát một mảng gồm 10 từ số nguyên trên ngăn xếp sau thanh ghi `$ra`.

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

Cấp phát mảng trong bộ nhớ heap

- Để cấp phát một mảng trên bộ nhớ heap, số lượng phần tử cần cấp phát được nhân với kích thước của từng phần tử để có được lượng bộ nhớ cần cấp phát.
- Ví dụ: Chương trình con cấp phát mảng trong bộ nhớ heap `AllocateArray`.

Chương trình con AllocateArray

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*

In một mảng

- Ví dụ cách truy cập các mảng bằng cách tạo một chương trình con `PrintIntArray` để in các phần tử trong một mảng số nguyên.
- Hai biến được truyền vào chương trình con, **\$a0** là địa chỉ cơ sở của mảng và **\$a1** là số phần tử cần in.
- Chương trình con xử lý mảng trong một vòng lặp bộ đếm và in ra từng phần tử theo sau bởi dấu `", "`.

In một mảng

- Mã giả như sau:

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

In một mảng

```
.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
```

In một mảng

```
.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
```

In một mảng

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:

In một mảng

```
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

- Sắp xếp là quá trình sắp xếp dữ liệu theo thứ tự tăng dần hoặc giảm dần.
- Ví dụ sau đây sẽ giới thiệu Bubble Sort, để sắp xếp dữ liệu số nguyên trong một mảng.

Bubble Sort


- Xét mảng sau đây chứa các giá trị nguyên.

55	27	13	5	44	17	36
-----------	-----------	-----------	----------	-----------	-----------	-----------

Bubble Sort

Việc sắp xếp được thực hiện trong hai vòng lặp.

- Vòng lặp bên trong đi qua một lần dữ liệu so sánh các phần tử trong mảng và hoán đổi chúng nếu chúng không theo đúng thứ tự.

27	13	5	44	17	36	55
						

Bubble Sort

- Quá trình này tiếp tục cho đến khi một vòng hoàn chỉnh được thực hiện. Ở cuối vòng lặp bên trong, giá trị lớn nhất của mảng nằm ở cuối mảng và ở đúng vị trí của nó. Mảng sẽ như sau.

27	13	5	44	17	36	55
----	----	---	----	----	----	----

Bubble Sort

Việc sắp xếp được thực hiện trong hai vòng lặp.

- Một vòng lặp bên ngoài hiện đang chạy lặp lại vòng lặp bên trong và giá trị lớn thứ hai di chuyển đến vị trí chính xác.

27	13	5	17	36	44	55
----	----	---	----	----	----	----

- Việc lặp lại vòng lặp bên ngoài này cho tất cả các phần tử dẫn đến mảng được sắp xếp theo thứ tự tăng dần.

Bubble Sort

- Mã giả

```
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 trong hợp ngữ MIPS

```
.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 trong hợp ngữ MIPS

```
# 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 trong hợp ngữ MIPS

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)
lw $s0, 4($sp)
lw $s1, 8($sp)
lw $s2, 12($sp)
lw $s3, 16($sp)
addi $sp, $sp, 20
jr $ra
```

#restore stack information

Bubble Sort trong hợp ngữ MIPS

```
# 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 element 2
# Side Effects:  Array is changed to swap element 1 and 2
```

Swap:

```
    sll $t0, $a1, 2      # calculate 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 trong hợp ngữ MIPS

```
# 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 trong hợp ngữ MIPS

```
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 trong hợp ngữ MIPS

Mars Messages

```
[,55,27,13,5,44,17,36]  
[,5,13,17,27,36,44,55]  
-- program is finished running --
```

Clear

Bài tập 10.1

- Thay đổi chương trình con `PrintIntArray` để nó in mảng từ phần tử cuối cùng đến phần tử đầu tiên.

Bài tập 10.2

- Đoạn mã giả sau đây chuyển đổi giá trị đầu vào của một số thập phân từ $1 \leq n \leq 15$ thành một chữ số hệ cơ số 16.
- Chuyển đoạn mã giả này sang hợp ngữ MIPS.

Bài tập 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]

}
```

Bài tập 10.3

- Thực hiện Bubble Sort với kích thước người dùng nhập vào là kích thước tối đa của mảng, sau đó điền vào mảng các số ngẫu nhiên.
- Sắp xếp mảng bằng cách sử dụng Bubble sort.
- In ra mảng.

Bài tập 10.4

Chương trình con `AllocateArray` không chính xác ở chỗ việc cấp phát bộ nhớ có thể nằm trên bất kỳ ranh giới nào. Đây là một vấn đề nếu mảng gồm các phần tử phải nằm trên một ranh giới cụ thể. Ví dụ: nếu mảng chứa các số nguyên phải nằm trên Ranh giới từ.

- Sử dụng chương trình con `PromptString` và `AllocateArray`, chỉ ra cách giải quyết vấn đề.
- Thay đổi chương trình `AllocateArray` để luôn thực hiện cấp phát trên ranh giới từ kép.

Kết thúc tuần 10