LAB 05: Arrays and Files

Saleh AlSaleh

salehs@kfupm.edu.sa

King Fahd University of Petroleum and Minerals
College of Computing and Mathematics
Computer Engineering Department

COE301: Computer Architecture
Term 222

Agenda

Static Allocation

- Allocates one variable or an array of variables in the static area of data segment.

Static Allocation

- Allocates one variable or an array of variables in the static area of data segment.
- Array size is determined at assemble time.

Static Allocation

- Allocates one variable or an array of variables in the static area of data segment.
- Array size is determined at assemble time.
- Data types: byte (8 bits), half-word (16 bits), word (32 bits).

Static Allocation

- Allocates one variable or an array of variables in the static area of data segment.
- Array size is determined at assemble time.
- Data types: byte (8 bits), half-word (16 bits), word (32 bits).
- Declaration and initialization at the same time example:

```
.data
secretbyte:   .byte    0xAB
secrethalf:   .half    0xABCD
secretword:   .word    0x89ABCDEF
```

Static Allocation

- Allocates one variable or an array of variables in the static area of data segment.
- Array size is determined at assemble time.
- Data types: byte (8 bits), half-word (16 bits), word (32 bits).
- Declaration and initialization at the same time example:

```
.data
secretbyte:   .byte   0xAB
secrethalf:   .half   0xABCD
secretword:   .word   0x89ABCDEF
```

```
.data
secretbytes:   .byte   0xAB:10
secrethalves:  .half   0:15
secretwords:   .word   1:20
```

Static Allocation

- Allocates one variable or an array of variables in the static area of data segment.
- Array size is determined at assemble time.
- Data types: byte (8 bits), half-word (16 bits), word (32 bits).
- Declaration and initialization at the same time example:

```
.data
secretbyte:   .byte   0xAB
secrethalf:   .half   0xABCD
secretword:   .word   0x89ABCDEF
```

```
.data
secretbytes:   .byte  0xAB:10
secrethalves:  .half  0:15
secretwords:   .word  1:20
```

- Declaration only example:

```
.data
secretarr: .space  100
```

Dynamic Allocation

- Allocates one or more bytes at run time in the dynamic area (heap) of the data segment.

Dynamic Allocation

- Allocates one or more bytes at run time in the dynamic area (heap) of the data segment.
- Some programs require different array sizes based on some inputs.

Dynamic Allocation

- Allocates one or more bytes at run time in the dynamic area (heap) of the data segment.
- Some programs require different array sizes based on some inputs.
- Use system call ($v0 = 9) and number of bytes to allocate in $a0.

Dynamic Allocation

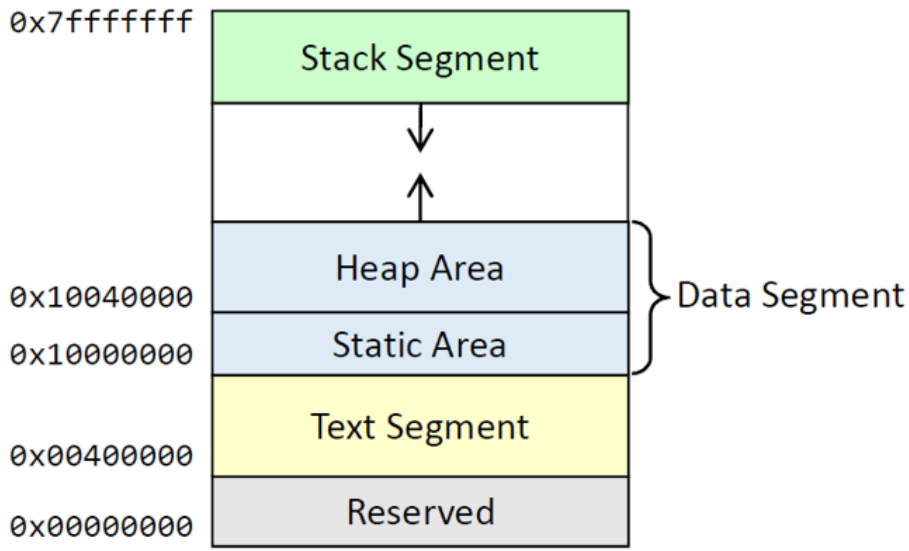
- Allocates one or more bytes at run time in the dynamic area (heap) of the data segment.
- Some programs require different array sizes based on some inputs.
- Use system call ($v0 = 9) and number of bytes to allocate in $a0.
- The base address will be returned in $v0, this base address needs to be saved.

Dynamic Allocation

- Allocates one or more bytes at run time in the dynamic area (heap) of the data segment.
- Some programs require different array sizes based on some inputs.
- Use system call ($v0 = 9) and number of bytes to allocate in $a0.
- The base address will be returned in $v0, this base address needs to be saved.
- Example:

```
li $v0, 9
li $a0, 30
syscall
# base address will be stored in $v0
```

Memory Organization



MIPS Memory Organization

Address Calculation

- 1-D Array Address Calculation
  - arr1D: .type 0:20 # e.g. int arr1D[20];

Address Calculation

- 1-D Array Address Calculation
  - arr1D: .type 0:20 # e.g. int arr1D[20];
  - Address of arr1D[i] =

Address Calculation

- 1-D Array Address Calculation
  - arr1D: .type 0:20 # e.g. int arr1D[20];
  - Address of arr1D[i] =
    base_address(arr1D) + (i * element_size)

Address Calculation

- 1-D Array Address Calculation
  - arr1D: .type 0:20 # e.g. int arr1D[20];
  - Address of arr1D[i] =
    base_address(arr1D) + (i * element_size)
- 2-D Array Address Calculation
  - arr2D: .type 0:20 # e.g. int arr1D[4][5];

Address Calculation

- 1-D Array Address Calculation
  - arr1D: .type 0:20 # e.g. int arr1D[20];
  - Address of arr1D[i] =
    base_address(arr1D) + (i * element_size)
- 2-D Array Address Calculation
  - arr2D: .type 0:20 # e.g. int arr1D[4][5];
  - Address of arr2D[i][j] =

Address Calculation

- 1-D Array Address Calculation
  - arr1D: .type 0:20 # e.g. int arr1D[20];
  - Address of arr1D[i] =
    base_address(arr1D) + (i * element_size)
- 2-D Array Address Calculation
  - arr2D: .type 0:20 # e.g. int arr1D[4][5];
  - Address of arr2D[i][j] =
    base_address(arr2D) + (i * col_size * element_size) + (j * element_size)

Files

- They provide an easy method to test applications that require many input and output values.

Files

- They provide an easy method to test applications that require many input and output values.
- For a file to be used, it needs to be **opened FIRST**.

Files

- They provide an easy method to test applications that require many input and output values.
- For a file to be used, it needs to be **opened FIRST**.
- System Call **13** is used to open a file with the following options:
  - $a0 address of null-terminated string containing the file name. Path can be relative to the location of MARS.jar file or an absolute path.
  - $a1 = 0 for read-only.
  - $a1 = 1 for write-only with truncate and create.
  - $a1 = 9 for write-only with create and append.
- It returns in $v0 a positive file descriptor if it can open the file or negative if error.
- File descriptor **NEEDS** to be saved for other system calls.

Files

- System Call **14** is used to read file contents with the following options:
  - $a0 = file descriptor
  - $a1 = address of input buffer
  - $a2 = maximum number of characters to read
- It returns in $v0 a positive number of characters read, zero if end of file or negative if error.

Files

- System Call **14** is used to read file contents with the following options:
  - $a0 = file descriptor
  - $a1 = address of input buffer
  - $a2 = maximum number of characters to read
- It returns in $v0 a positive number of characters read, zero if end of file or negative if error.
- System Call **15** is used to write contents to file with the following options:
  - $a0 = file descriptor
  - $a1 = address of output buffer
  - $a2 = number of characters to write
- It returns in $v0 a positive number of characters written or negative if error.

Files

- System Call **14** is used to read file contents with the following options:
  - $a0 = file descriptor
  - $a1 = address of input buffer
  - $a2 = maximum number of characters to read
- It returns in $v0 a positive number of characters read, zero if end of file or negative if error.
- System Call **15** is used to write contents to file with the following options:
  - $a0 = file descriptor
  - $a1 = address of output buffer
  - $a2 = number of characters to write
- It returns in $v0 a positive number of characters written or negative if error.
- System Call **16** is used to close file with $a0 containing the file descriptor.

Live Examples

Task #1

Write a MIPS assembly program that reads the size (n) of the message from the user. Then, the program allocates (n+1) bytes in the heap. After that, read a string of n characters from the user he/she wishes to encrypt. Next, read an encrypting key (e) from the user [1, 25]. Encrypt the original string with the encryption key using the following code. Finally, print out the encrypted string.

Sample Run

```
Enter n: 11
Enter string: Hello World
Enter e: 13
Encrypted string = Uryyb Jbeyq
```

```
for(i=0;i<n;i++){
  ch = str[i];
  if(isupper(ch)){
    ch = ch + e;
    if (ch > 0x5A)
      ch = ch - 26;
    }
  else if(islower(ch)){
    ch = ch + e;
    if (ch > 0x7A)
      ch = ch - 26;
    }
  str[i] = ch;
}
```

Caesar Encryption Algorithm

Task #2

Write a MIPS assembly program that asks the user for file name (max 50 chars). Open the file for reading. Next, read the file contents as characters (max 100 chars). After that, loop over each character, if the character is a digit (i.e. '0' to '9'), convert it to integer and store it in another array called "array_int". Assume the maximum number of integers in the file is 20. Finally, print "array_int" in reverse order.

Static data segment

```
.data
filename: .space 50
filecontents: .space 100
array_int: .word 0:20
```

Sample File (numbers.txt)

```
1 2 3 4 5 6
7 9 hello, world
851
```

Sample Run

```
Enter filename: numbers.txt
Integer array reversed = 1 5 8 9 7 6 5 4 3 2 1
```