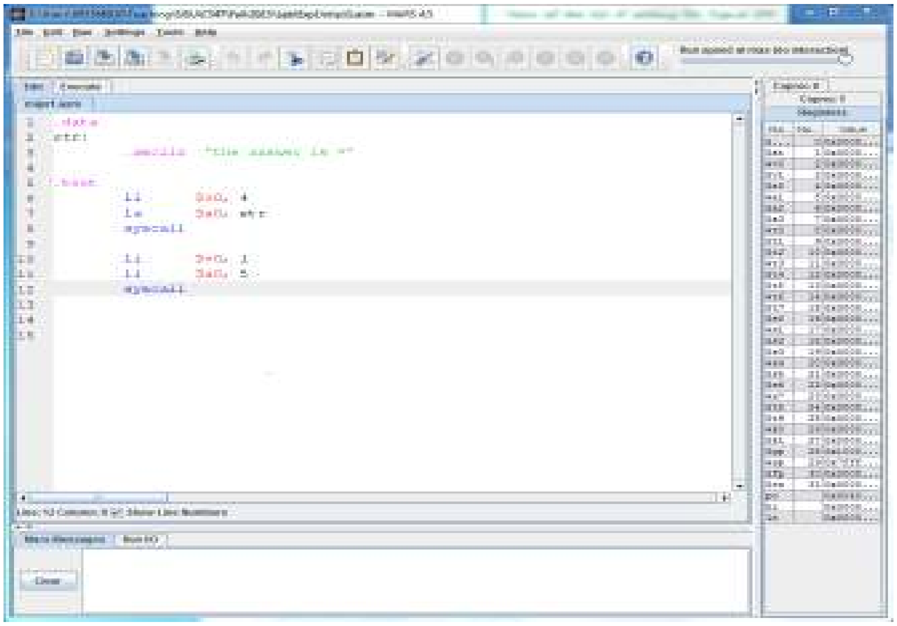
CS47 - Lecture 06

Kaushik Patra
(kaushik.patra@sjsu.edu)

Simulators

- Simulator virtualizes real life operations.
  - Flight simulator virtualizes flight experience for pilots.
- For computer organization & system class we'll use MIPS simulator – SPIM.
  - Simulates MIPS processor based computing environment.
  - We can program in assembly and then run the program on the virtual platform to observe processor and system behavior.

MARS IDE



```
.data
str:
        .asciiz "the answer is ="

.text
        li      $v0, 4
        la      $a0, str
        syscall

        li      $v0, 1
        li      $a0, 5
        syscall
```

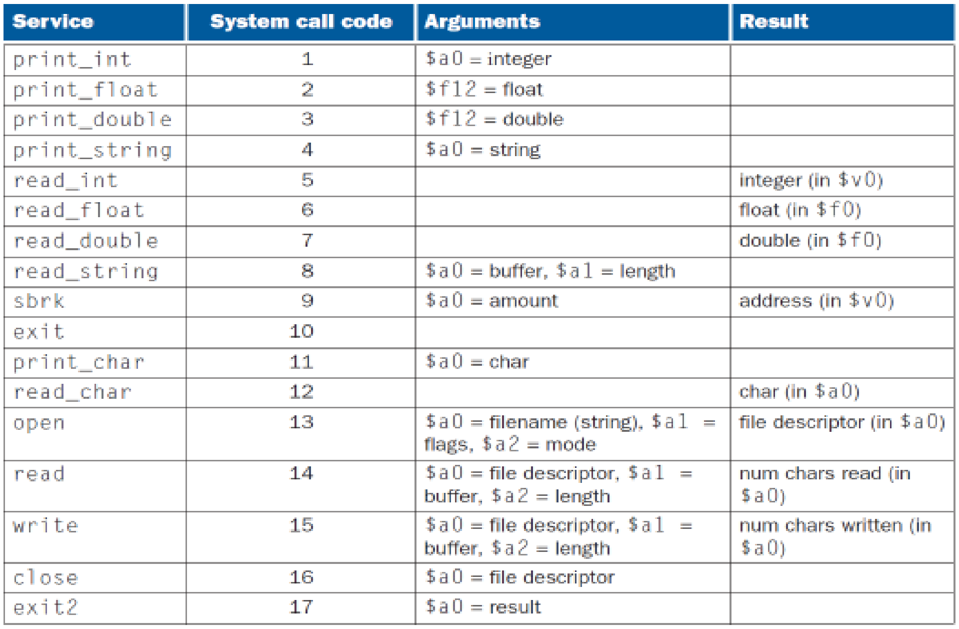
Objective

- Write a simple program which will print 'the answer is = 5'.
  - Get familiar with the system call
  - Run and step through the program
  - Observe data / text memory content
  - Observe register value changes
  - Use macro to wrap the following common codes.
    - Print integer, float, double, string
    - Read integer, float, double, string

System Calls

- SPIM provides a small set of OS like system call.
  - Standard input/output
  - File operations.
  - Memory allocation
  - Quitting from program

System Calls

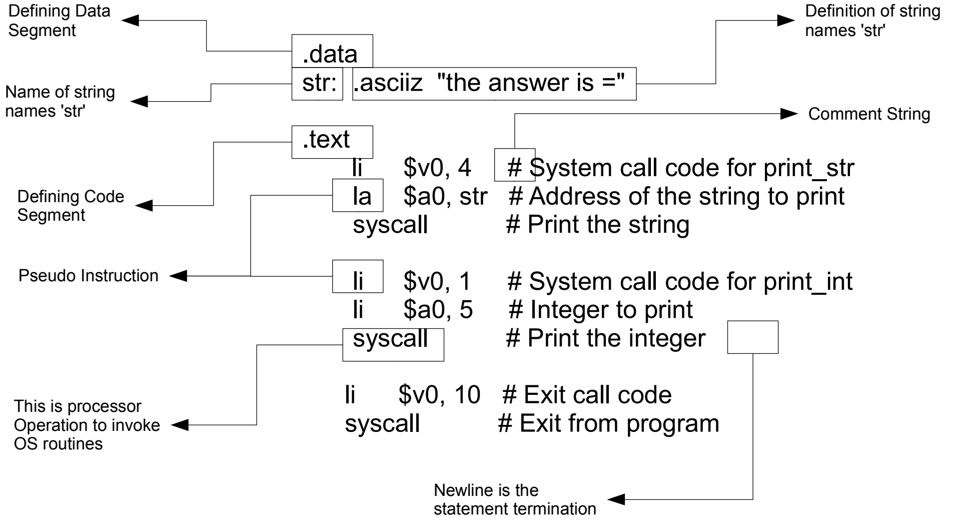


Service	System call code	Arguments	Result
`print_int`	1	`$a0` = integer	
`print_float`	2	`$f12` = float	
`print_double`	3	`$f12` = double	
`print_string`	4	`$a0` = string	
`read_int`	5		integer (in `$v0`)
`read_float`	6		float (in `$f0`)
`read_double`	7		double (in `$f0`)
`read_string`	8	`$a0` = buffer, `$a1` = length	
`sbrk`	9	`$a0` = amount	address (in `$v0`)
`exit`	10		
`print_char`	11	`$a0` = char	
`read_char`	12		char (in `$a0`)
`open`	13	`$a0` = filename (string), `$a1` = flags, `$a2` = mode	file descriptor (in `$a0`)
`read`	14	`$a0` = file descriptor, `$a1` = buffer, `$a2` = length	num chars read (in `$a0`)
`write`	15	`$a0` = file descriptor, `$a1` = buffer, `$a2` = length	num chars written (in `$a0`)
`close`	16	`$a0` = file descriptor	
`exit2`	17	`$a0` = result	

FIGURE A.9.1 System services.

First Program ...

- Write a code name exp1.asm as following.



```
.data
str:  .asciiz  "the answer is ="
.text
      li    $v0, 4    # System call code for print_str
      la    $a0, str  # Address of the string to print
      syscall         # Print the string

      li    $v0, 1    # System call code for print_int
      li    $a0, 5    # Integer to print
      syscall         # Print the integer

     li     $v0, 10   # Exit call code
     syscall          # Exit from program
```

Defining Data Segment

Name of string names 'str'

Definition of string names 'str'

Comment String

Defining Code Segment

Pseudo Instruction

This is processor Operation to invoke OS routines

Newline is the statement termination

Try a Macro ...

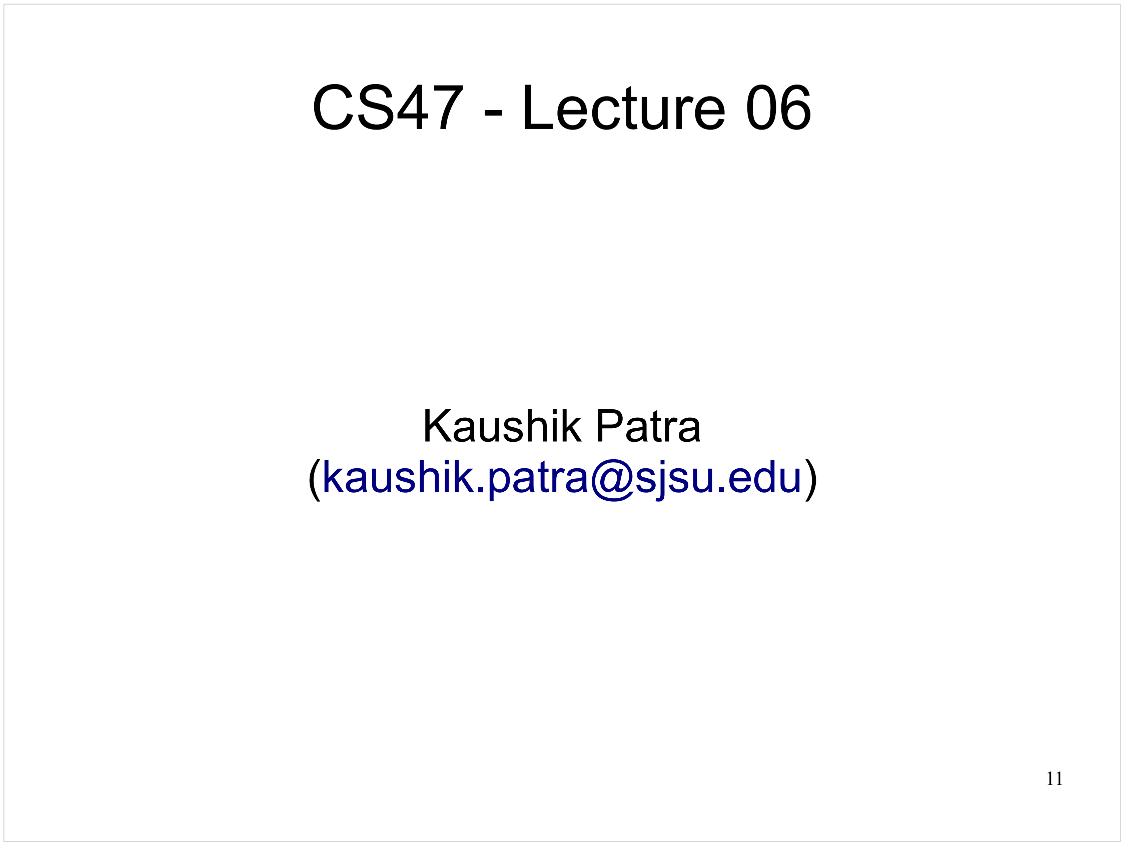
- Copy the code from exp1.asm to exp2.asm
- Create following macros and use it in exp2.asm
  - print_int
  - print_str
  - exit
- Label start of program as 'main'
  - Make this main as global using .globl
  - Also turn on in Mars → Setting → Initialize program counter to global main if available.

Try a Macro … Solution

```
#<------------------ MACRO DEFINITIONS ---------------------->#
     # Macro : print_str
     # Usage: print_str(<address of the string>)
     .macro print_str($arg)
     li   $v0, 4     # System call code for print_str
     la   $a0, $arg   # Address of the string to print
     syscall            # Print the string
     .end_macro

     # Macro : print_int
     # Usage: print_int(<val>)
     .macro print_int($arg)
     li   $v0, 1     # System call code for print_int
     li   $a0, $arg  # Integer to print
     syscall            # Print the integer
     .end_macro

     # Macro : exit
     # Usage: exit
     .macro exit
     li   $v0, 10
     syscall
     .end_macro

#<------------------ APPLICATION PROGRAM---------------------->#
#<------------------ DATA SEGMENT DEFINITION------------------>#
.data
str: .asciiz  "the answer is ="

#<------------------ CODE SEGMENT DEFINITION------------------>#
.text
.globl main
Main:     print_str(str)
          print_int(5)
          exit
```

Try a Macro ...

- Create cs47_macro.asm and put all the macro definition in that file.
- Copy the main code to exp3.asm and use the following to include the macro definition.
  - .include “cs47_macro.asm”
- Expand the macro definitions for other system calls
  - print_float, print_double
  - read_int, read_float, read_double, read_str
    - Let it take arguments to which reg the data to be read
    - Use pseudo instruction move.



CS47 - Lecture 06

Kaushik Patra
(kaushik.patra@sjsu.edu)

11