Prepared by Julian García, based on material by
David Albrecht and María García de la Banda

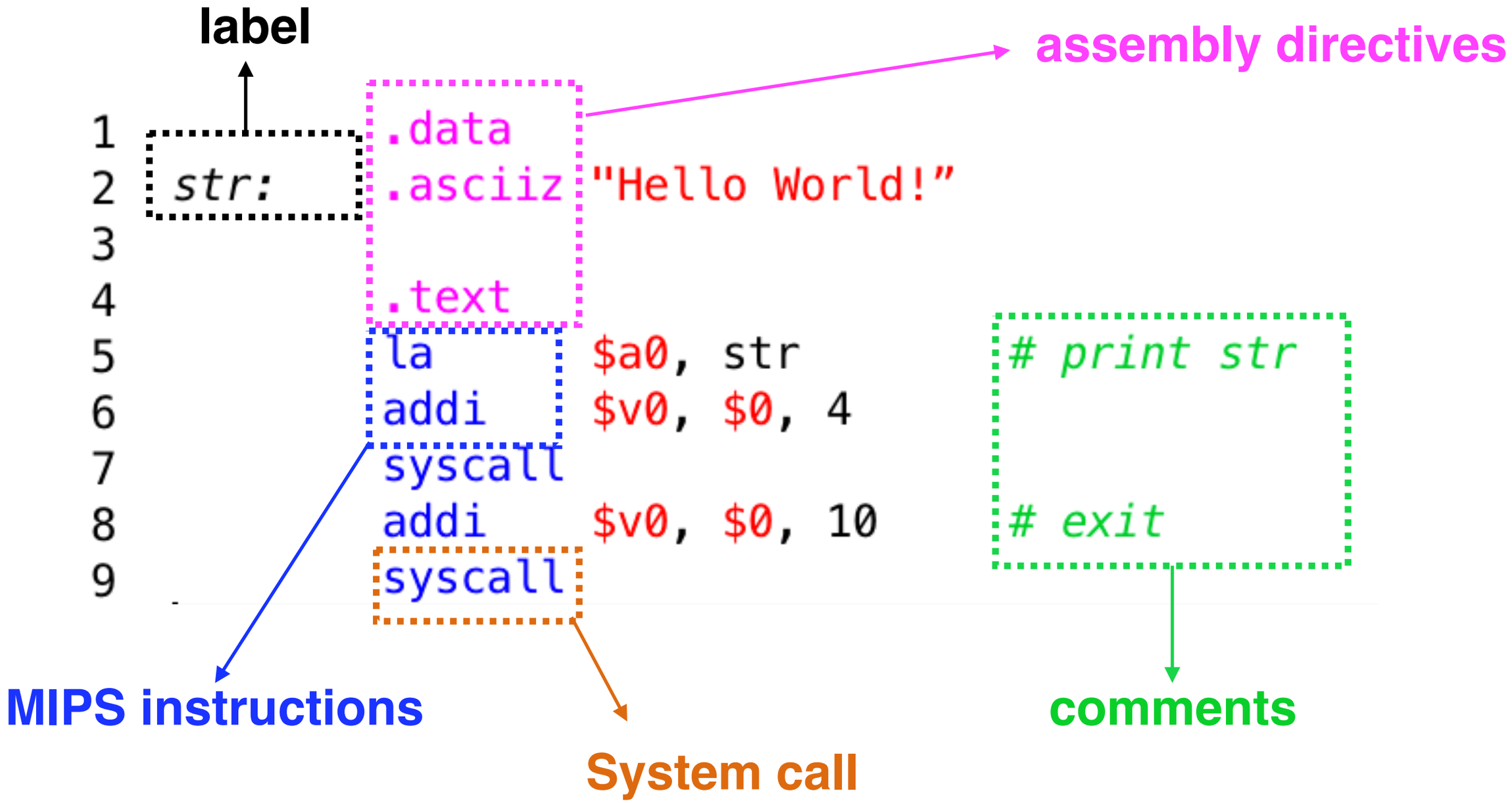
# Lecture 3
# Assembly Programming

## FIT 1008
## Introduction to Computer Science

# Objectives

- To understand how to **write simple MIPS programs**
- To understand how to read and write integers and strings in MIPS.
- To understand how to use the data segment.
- To understand some assembly directives.

# Hello World! (MIPS)



# General Purpose Registers

| Register name | Register number | Typical use |
|---|---|---|
| $zero | $0 | constant zero, cannot change, read only |
| $v0, $v1 | $2, $3 | function return values; system call number |
| $a0 - $a3 | $4 - $7 | function and system call arguments |
| $t0 - $t7, $t8, $t9 | $8 - $15, $24, $25 | temporary storage (caller-saved) |
| $s0 - $s7 | $16 - $23 | temporary storage (callee-saved) |
| $sp | $29 | top-of-stack pointer |
| $fp | $30 | stack frame pointer |
| $ra | $31 | function return address |

# Assembler Directives

- Always start with . (dot)
- Instruct the assembler to allocate space or data, or switch modes.
- Assembler directives don't assemble to machine language instructions

# Assembler Directives – Switch Mode

```
1           .data
2   str:    .asciiz "Hello World!"
3
4           .text
5           la      $a0, str        # print str
6           addi    $v0, $0, 4
7           syscall
8           addi    $v0, $0, 10     # exit
9           syscall
```

## .data

- Tells the assembler it is working in the data segment
- Remember: this is where **global variables** are

## .text

- Tells the assembler it is working in the text segment
- Remember: this is where the **code instructions** go

```
1               .data
2   str:        .asciiz "Hello World!"
3
4               .text
5               la      $a0, str            # print str
6               addi    $v0, $0, 4
7               syscall
8               addi    $v0, $0, 10         # exit
9               syscall
```

# Assembler Directives
# Allocate Space

Allocates memory in the data segment

- **`.space N`**
  allocate N bytes, store nothing

- **`.word w1 [, w2, w3, …]:`**
  allocate 4-byte words and store values in them

- **`.asciiz "string"`**
  allocates the string as a sequence of ASCII values, terminated by a zero byte (null character indicating end of string)

# Labels and symbols

- **Labels**: **names** for **locations in memory**
- But they need to be translated into numbers before execution
- The assembler uses a **symbol table** to help it do this
  - When it sees a label being **defined**:
    It puts the label name and the current address in the table
  - When it sees a label being **used**:
    It looks the name up in the table to find what address it refers to

# Input/Output

- I/O is complicated. Usually handled by the Operating System (OS)

- MIPS programs request I/O from the OS using a special command called **syscall**

# System Services

To make a system call

1. Work out which service you want
2. Put service's call code in register $v0
3. Put argument (if any) in registers $a0, $a1
4. Perform the syscall instruction
5. Result (if any) will be returned in register $v0

| Service | Call | Argument | Result |
|---|---|---|---|
| Print integer | 1 | $a0 (int to be printed) | n/a |
| Print string | 4 | $a0 (addr of first char of string) | n/a |
| Read integer | 5 | n/a | $v0 (integer) |
| Read string | 8 | $a0 (addr to put string)<br>$a1 (number of bytes to read) | n/a |
| Allocate | 9 | $a0 (number of bytes requested) | $v0 (addr of allocated |
| Exit program | 10 | n/a | n/a |

```
1               .data
2   str:        .asciiz "Hello World!"
3
4               .text
5               la      $a0, str            # print str
6               addi    $v0, $0, 4
7               syscall
8               addi    $v0, $0, 10         # exit
9               syscall
```

# hello_john.py

```
name = input('Enter name (max 60 chars):  ')

print('Hello ' + name)
```

```
1           .data
2   prompt: .asciiz "Enter name (max 60 chars):  "
3   str:    .asciiz "Hello "
4   name:   .space 60
5
6           .text
7           la      $a0, prompt     # print prompt
8           addi    $v0, $0, 4
9           syscall
10
11          la      $a0, name       # read name
12          addi    $a1, $0, 60
13          addi    $v0, $0, 8
14          syscall
15
16          la      $a0, str        # print str
17          addi    $v0, $0, 4
18          syscall
19
20          la      $a0, name       # print name
21          addi    $v0, $0, 4
22          syscall
23
24          addi    $v0, $0, 10     # exit
25          syscall
```

# convert_temp.py

```
temp_C = int(input('Enter temperature in Celsius '))

temp_F = int(9*temp_C/5 + 32)

print('Temperature in Fahrenheit is ' + str(temp_F))
```

**Suggestion**: write first in **simple** python

# convert_temp.py

```
temp_C = int(input('Enter temperature in Celsius '))

temp_F = int(9*temp_C/5 + 32)

print('Temperature in Fahrenheit is ' + str(temp_F))
```

1. Setup up string constants and global variables
2. Read temp_C
3. Compute temp_F
4. Print temp_C

# Setup constant strings and global variables

```
1           .data
2   prompt: .asciiz "Enter temperature in Celsius:"
3   str:    .asciiz  "Temperature in Fahrenheit is"
4   temp_C: .word    0
5   temp_F: .word    0
```

# Read temp_C

```
6   .text
7   la      $a0, prompt      # print prompt
8   addi    $v0, $0, 4
9   syscall
10  addi    $v0, $0, 5       # read int
11  syscall
12  sw      $v0, temp_C
```

**sw:** store what is in $vo, in address temp_C

| | | | | |
|---|---|---|---|---|
| sb Rsrc2, Addr (or label $\star$) | Store byte | mem8[Addr] = Rsrc2 | - | - |
| sh Rsrc2, Addr (or label $\star$) | Store halfword | mem16[Addr] = Rsrc2 | - | - |
| sw Rsrc2, Addr (or label $\star$) | Store word | mem32[Addr] = Rsrc2 | - | - |

# Arithmetic Instructions

- **addition** (+)

```
add $t0, $t1, $t2   # $t0 = $t1 + $t2
```

- addi – **immediate addition** (+)

```
addi $t0, $t1, 5    # $t0 = $t1 + 5
```

- **subtraction** (–)

```
sub $t0, $t1, $t2   # $t0 = $t1 - $t2
```

- **multiplication** (*)

```
mult $t1, $t2    # LO=$t1*$t2,
                 # HI=overflow
```

- **division** (/)

```
div $t1, $t2   # LO=$t1/$t2,
               # HI=remainder
```

# Data Movement Instructions

- move from HI
  - `mfhi $t0  # $t0 = HI`
- move from LO
  - `mflo $t0  # $t0 = LO`

# Compute temp_F

```
lw      $t0, temp_C
addi    $t1, $0, 9
mult    $t0, $t1
mflo    $t0
addi    $t1, $0, 5
div     $t0, $t1
mflo    $t0
addi    $t1, $0, 32
add     $t0, $t0, $t1
sw      $t0, temp_F
```

temp_F = **int**(9*temp_C/5 + 32)

# Print temp_F

```
la      $a0, str
addi    $v0, $0, 4
syscall
lw      $a0, temp_F
addi    $v0, $0, 1
syscall
addi    $v0, $0, 10
syscall
```

# Summary

- Simple MIPS Programming
  - String handling
  - Arithmetic
  - Input/Output: system calls
  - Data Segment
  - Text Segment
  - Assembler directives