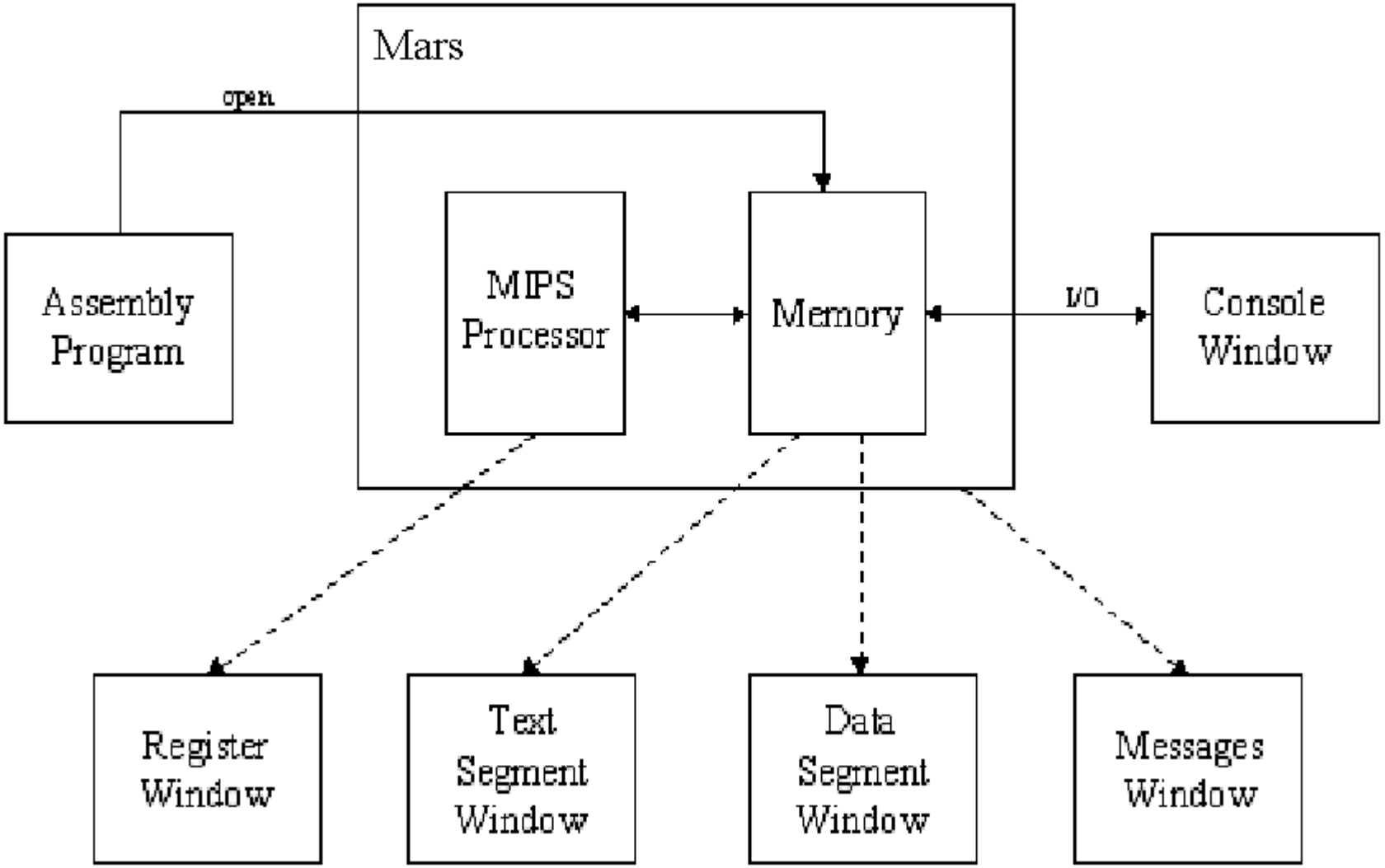
COMP2611: Computer Organization

Introduction to MARS & MIPS syscall

Introduction to MARS

- MARS is a MIPS computer simulator.
- It can execute MIPS assembly programs by emulating itself as an actual MIPS computer.
- It provides some, but not all, operating system services which you will see later.

The architecture of MARS

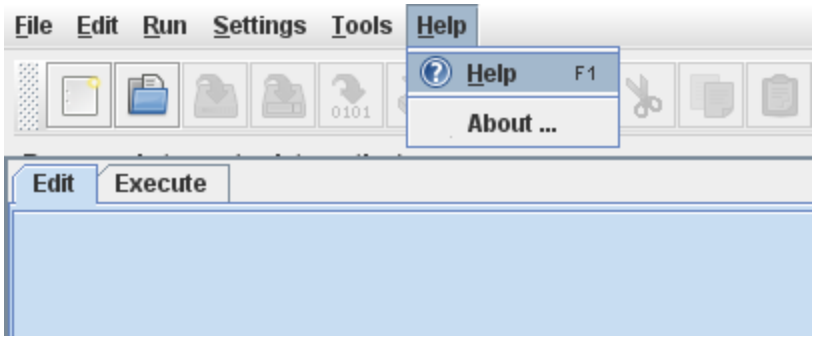


Getting and installing JRE

- Before running MARS, you need Java Runtime Environment (JRE) of Java SE 5 (also called Java 1.5) or later installed. It is already done in the lab room.
  - You can choose the version of JRE to download on this website http://www.oracle.com/technetwork/java/javase/downloads/index.html
  - Note that even if you use 64-bit Windows, you can still download and install 32-bit (not only 64-bit) version of JRE on your Windows.
  - To install JRE, double-click or run the downloaded file and follow its installation instructions.

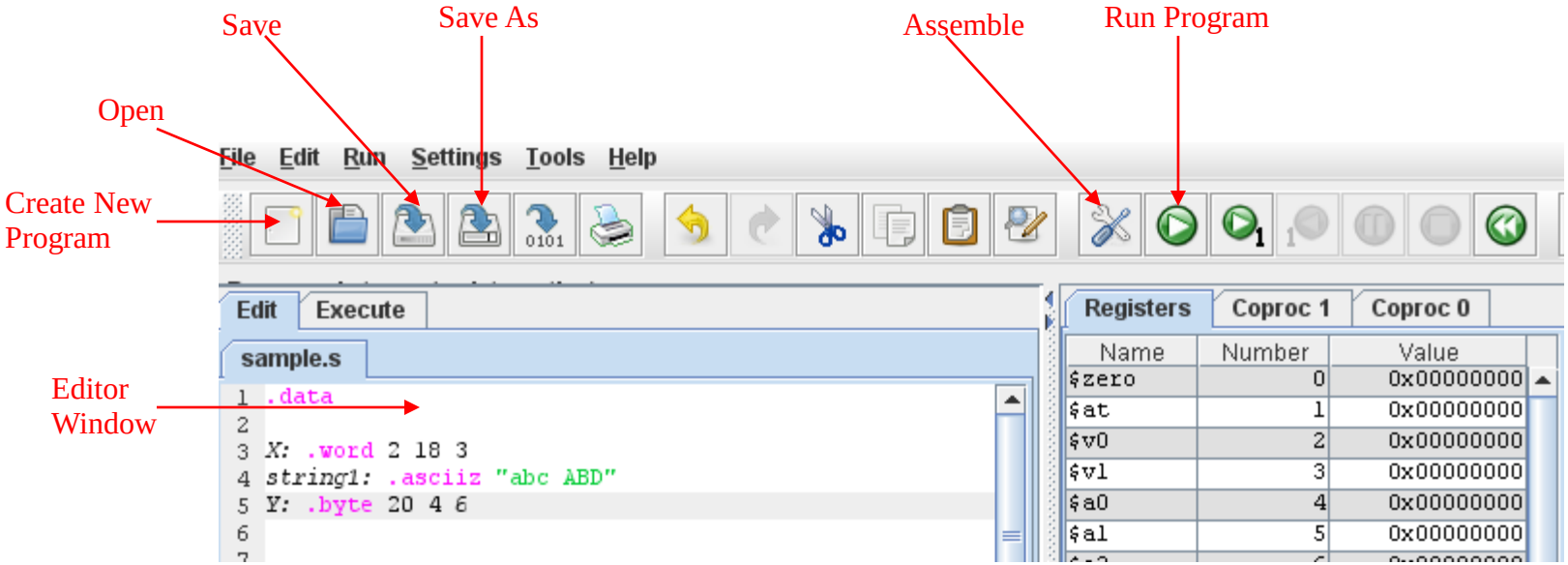
Getting and running MARS

- To get MARS and run it
  - Browse the official site http://courses.missouristate.edu/KenVollmar/MARS/
  - Follow the instruction there (e.g., on the Download section) to download and run MARS.
  - You can just download MARS from http://course.cse.ust.hk/comp2611/# , too. Then double-click the downloaded .jar file in Windows to run MARS.
- The Help manual of using MARS can viewed by selecting the Help->Help menu command on MARS.



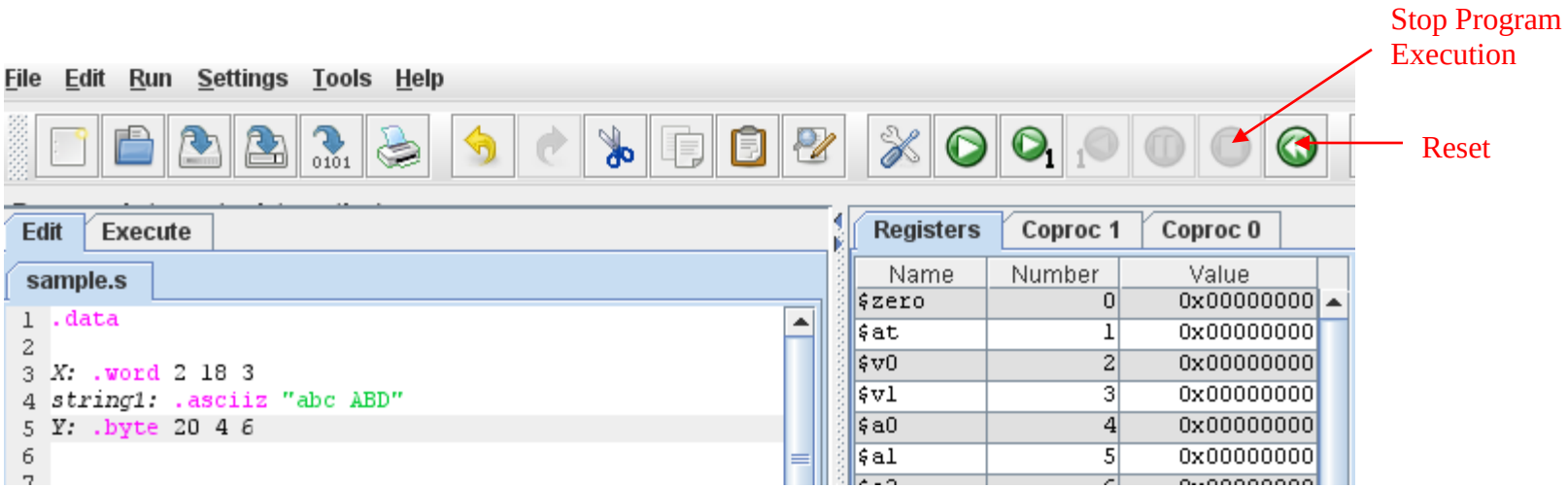
Running an assembly program

- To run an assembly program
  - **Create** a new program file on MARS.
  - Write its program code on the Editor window.
  - **Save** or **Save As** the file with ".s" as the file extension. Note that you can also **Open** an existing .s file on MARS, instead of creating a new file.
  - Then **Assemble** the program file.
  - Finally, **Run** it.



Stopping a program execution

- After the program execution runs past the last instruction of the program, it will terminate normally.
- During the execution, it can also be terminated immediately using the **Stop** button.
- After the execution is terminated (in any ways), it can be reset (all the registers and memory are re-initialized) using the **Reset** button for another fresh start of the execution.
- Some other buttons are for debugging a program and will be taught in a future lab.

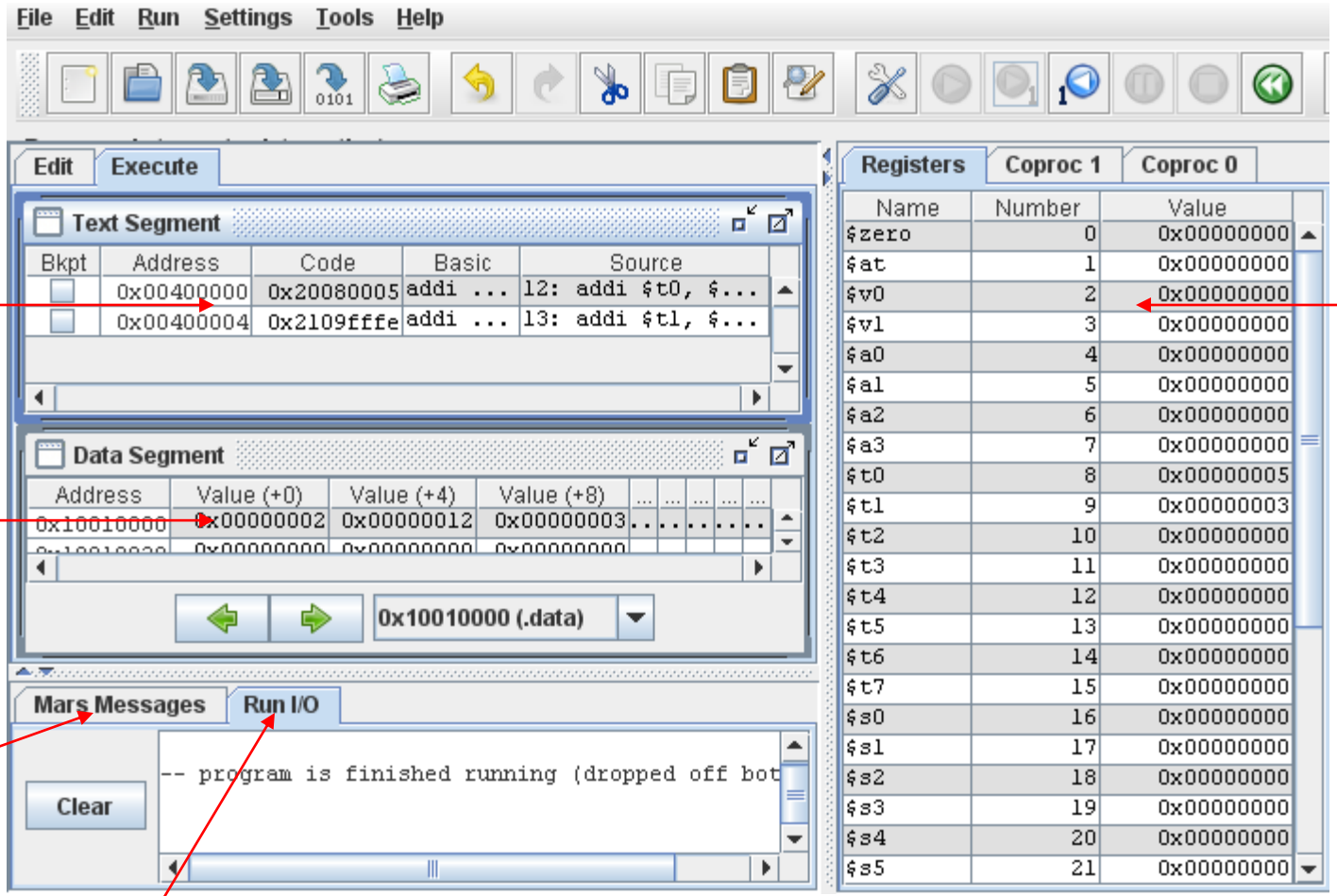


Example program

Try to create and run the following example program on MARS:

```
.data

X:          .word 2 18 3
Y:          .word 20 4

.text
.globl __start
__start:

addi $t0, $zero, 5
addi $t1, $t0, -2
```

MARS user interface



Text Segment Window

Data Segment Window

Messages Window

Console I/O Window

Registers Window

Registers Window

- **Registers Window**
  - displays the registers of a MIPS processor.
  - including
    - the 32 general-purpose registers
  - By default, a register value is displayed in hexadecimal format using 2's complement.

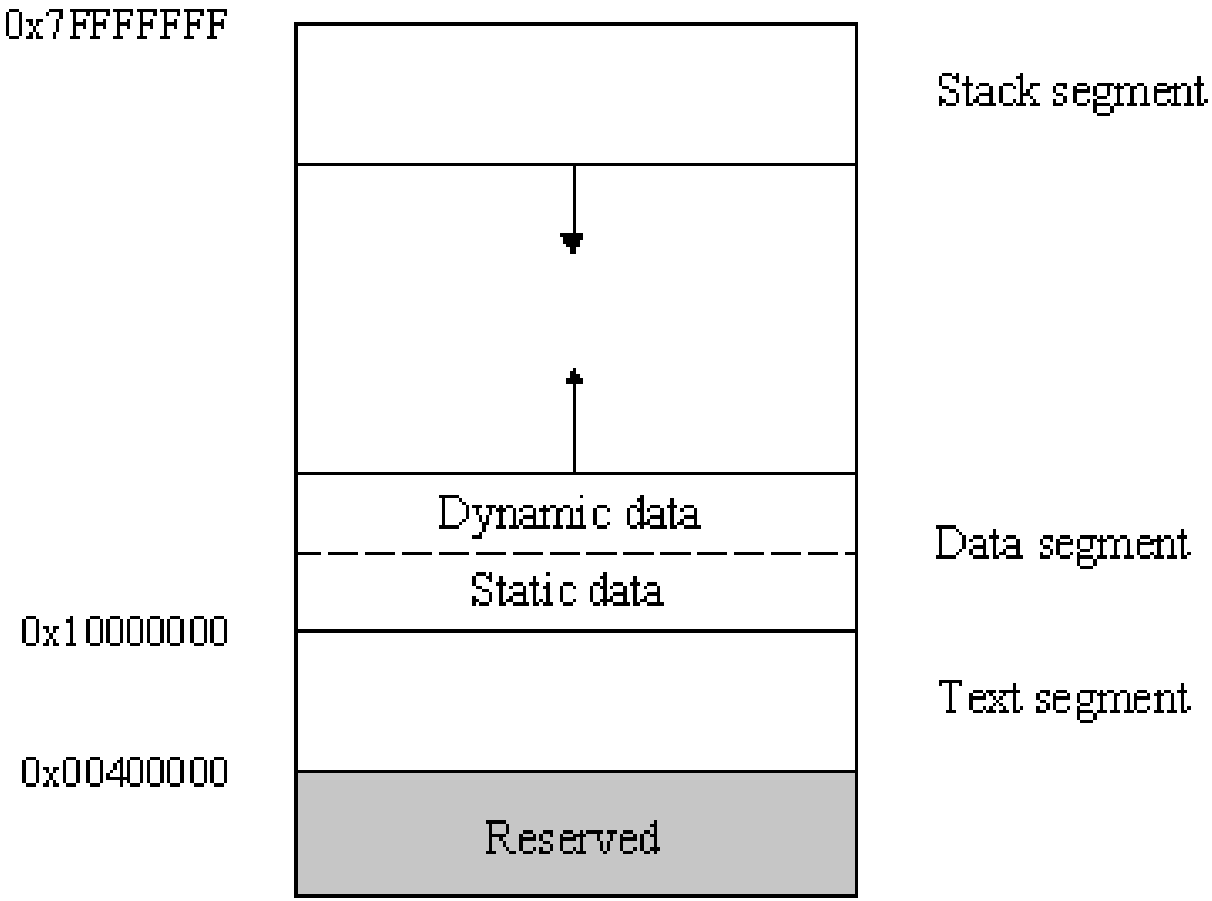
Registers Window

- After running the example program you just created,
  - examine how the values of the registers t0 and t1 on the Registers Window correspond to the program code;
  - modify the program code to set the value of t0 to 1 instead of 5 (as shown below) and save the code;
  - assemble and run the modified program.
- What are the values of the registers t0 and t1 in the Registers Window?

```
.
.
.

__start:

addi $t0, $zero, 1
addi $t1, $t0, -2

.
.
.
```

MIPS program memory layout

The layout of memory



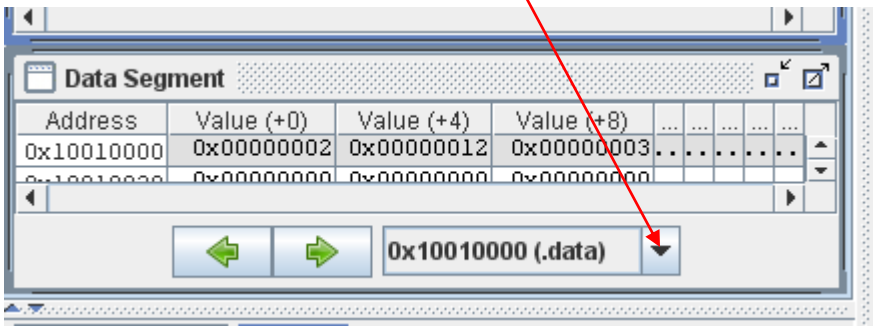
Text Segment Window

- **Text Segment Window**
  - displays the TEXT segment of the memory contents,

    i.e. the instruction code in the .text segment of the program.
  - By default, your program code begins at 0x00400000.
  - Due to the 32-bit nature of MIPS, the second instruction is located at 0x00400004.
- Examine how the Text Segment Window reflects the instructions in the modified example program.

Data Segment Window

- **Data Segment Window**
  - displays various parts of the memory of your MIPS program, e.g., DATA, STACK, etc.
  - The data defined in the **.data** segment of the program is stored in the DATA part of the memory.
  - This **Drop-down List** button can be clicked to select the different part of memory for the display.
  - The data on the window is updated as the program executes.
  - By default, a memory value is displayed in hexadecimal format using 2's complement.
- How is the data in the example program displayed in the window?



Messages Window & Console I/O Window

- **Messages Window**
  - displays messages from the MIPS simulator of MARS.
  - It does not display outputs from an executing program.

- **Console I/O Window**
  - When a program reads or writes, its IO appears on this window.

MIPS syscall services

- A MIPS instruction syscall is defined to perform a system service, e.g., Console Input/Output.
- Run the example program printString.s which uses the syscall to print the string "Hello World" to the console.
- Before executing the syscall instruction, you need to:
  - store the *system call code* (an integer) in the register v0, and the service performed by the syscall is determined by this register value (at the moment of executing the syscall instruction) .
  - pass any argument(s) for the syscall service via some particular register(s), e.g., passing the output value in the register a0 for printing an integer to the console.

Common syscall services

- Some common syscall services (you must know the yellow ones):

Service	System Call Code ($v0)	Arguments	Result	Example
print_int	**1**	**$a0=integer**		**li $v0, 1 li $a0, 100 syscall**
print_float	2	$f12=float		
print_double	3	$f12=double		
print_string	**4**	**$a0=start address of the string**		
read_int	**5**		**integer (in $v0)**	**li $v0, 5 syscall # $v0 = input value**
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**			**li $v0, 10 syscall**

Printing a string to console

In C++	In MIPS

// C++ version
// declare the string mesg
char mesg[] =
{'H', 'e', 'l', 'l', 'o', ' ',
'W', 'o', 'r', 'l', 'd', '\n', '\0' };
 	
// main is the default
//starting point of the program
void main() {
 	
cout << mesg;
 	
}
	#------ Data Segment ---------
	.data
	# declare the string mesg
	mesg: .asciiz "Hello World\n"
	
	#------ Text Segment ---------
	.text
	
	.globl main
	main:
	
	# Execute the "print_str" system call
	li $v0, 4
	la $a0, mesg
	syscall

Address	
Mesg	'H'
mesg+1	'e'
mesg+2	'l'
mesg+3	'l'
mesg+4	'o'
mesg+5	' '
mesg+6	'W'
mesg+7	'o'
mesg+8	'r'
mesg+9	'l'
mesg+10	'd'
mesg+11	'\n'
mesg+12	'\0'

Printing a string to console

In C++	In MIPS

// C++ version
// declare the string mesg
char mesg[] =
{'H', 'e', 'l', 'l', 'o', ' ',
'W', 'o', 'r', 'l', 'd', '\n', '\0' };
 	
// main is the default
//starting point of the program
void main() {
 	
cout << mesg;
 	
}
	#------ Data Segment ---------
	.data
	# declare the string mesg
	mesg: .asciiz "Hello World\n"
	
	#------ Text Segment
	.text
	
	.globl main
	main:
	
	# Execute the "print_str" system call
	li $v0, 4
	la $a0, mesg
	syscall

Setting v0 to 4, the processor knows we need to print a string to the console when executing a syscall.

Address	
Mesg	'H'
mesg+1	'e'
mesg+2	'l'
mesg+3	'l'
mesg+4	'o'
mesg+5	' '
mesg+6	'W'
mesg+7	'o'
mesg+8	'r'
mesg+9	'l'
mesg+10	'd'
mesg+11	'\n'
mesg+12	'\0'

Printing a string to console

In C++	In MIPS

// C++ version
// declare the string mesg
char mesg[] =
{'H', 'e', 'l', 'l', 'o', ' ',
'W', 'o', 'r', 'l', 'd', '\n', '\0' };
 	
// main is the default
//starting point of the program
v
}
 	
When **la $a0, mesg** is executed, the starting address of the string will be assigned to the register a0.
	#------ Data Segment ---------
	.data
	# declare the string mesg
	mesg: .asciiz "Hello World\n"
	
	#------ Text Segment
	.text
	
	.globl mair
	main:
	
	# Execute the "print_str" system call
	li $v0, 4
	la $a0, mesg
	syscall
	
	Setting v0 to 4, the processor knows we need to print a string to the console when executing a syscall.

Address	
Mesg	'H'
mesg+1	'e'
mesg+2	'l'
mesg+3	'l'
mesg+4	'o'
mesg+5	' '
mesg+6	'W'
mesg+7	'o'
mesg+8	'r'
mesg+9	'l'
mesg+10	'd'
mesg+11	'\n'
mesg+12	'\0'

Printing a string to console

In C++

```
// C++ version
// declare the string mesg
char mesg[] =
   {'H', 'e', 'l', 'l', 'o', ' ',
    'W', 'o', 'r', 'l', 'd', '\n', '\0' };

// main is the default
//starting point of the program
v
}
```

In MIPS

```
#------ D
       .da
# declare the string mesg
mesg:   .asciiz "Hello World\n"

#------ Text Segment
       .text

       .globl mair
main:

# Execute the "print_str" system call
       li $v0, 4
       la $a0, mesg
       syscall
```

e.g., if mesg (character 'H') is located at the 1001-th byte of memory, then a0 = 1001.

When **la $a0, mesg** is executed, the starting address of the string will be assigned to the register a0.

Setting v0 to 4, the processor knows we need to print a string to the console when executing a syscall.

Address	
Mesg	'H'
mesg+1	'e'
mesg+2	'l'
mesg+3	'l'
mesg+4	'o'
mesg+5	' '
mesg+6	'W'
mesg+7	'o'
mesg+8	'r'
mesg+9	'l'
mesg+10	'd'
mesg+11	'\n'
mesg+12	'\0'

Printing a string to console

In C++

```
// C++ version
// declare the string mesg
char mesg[] =
   {'H', 'e', 'l', 'l', 'o', ' ',
    'W', 'o', 'r', 'l', 'd', '\n', '\0' };

// main is the default
//starting point of the program
v
}
```

In MIPS

```
#------ D
       .da
# declare the string mesg
mesg:   .asciiz "Hello World\n"

#------ Text Segment
       .text

       .globl mair
main:

# Execute the "p
       li $v0, 4
       la $a0, me
       syscall
```

e.g., if mesg (character 'H') is located at the 1001-th byte of memory, then a0 = 1001.

Address	
Mesg	'H'
mesg+1	'e'
mesg+2	'l'
mesg+3	'l'
mesg+4	'o'
mesg+5	' '
mesg+6	'W'
mesg+7	'o'
mesg+8	'r'
mesg+9	'l'
mesg+10	'd'
mesg+11	'\n'
mesg+12	'\0'

When **la $a0, mesg** is executed, the starting address of the string will be assigned to the register a0.

Setting v0 to 4, the processor knows we need to print a string to the console when executing a syscall.

After executing syscall, the processor **reads the memory byte by byte** from the address in a0 (e.g. 1001--> 1002 --> 1003 ... and so on). The corresponding **character** will be **displayed one by one until the end of string character ('\0') is read.**

Example programs

- Try the following example programs:
  - printString.s (for Printing a string to the console).
  - printInt.s (for Printing an integer to the console).
  - readInt.s (for Reading an integer from the console).

Syscall service "exit"

- The syscall service "exit" terminates the program immediately after this syscall instruction is executed.

```
# starting main program
.text
.globl __start
__start:

addi $t0, $zero, 5
addi $t1, $t0, -2

li $v0, 10
syscall     # the program is terminated after executing this syscall

# the codes below will never be executed
addi $t1, $t1, 1
add $t1, $t0, $t1
```

- Try the example programs exitExample1.s and exitExample2.s.

Example program

- Try the example program combinedSyscalls.s:
  - It demonstrates the use of various syscall services together.
  - It prompts the user to enter two numbers on the console, reads the input numbers and prints their sum to the console.

Exercise

- By using the syscall services you have learnt:
  - write a MIPS program that prompts the user for two integer inputs,
  - and displays the sum of the two integers,
  - the program should be able to exit using the syscall service after displaying the sum,
  - you do not need to verify the correctness of the input integers.

Conclusion

- You have:
  - learnt how to get and use MARS;
  - learnt how to create and execute a MIPS program in MARS;
  - learnt using the user interface of MARS;
  - how to perform a system service using the instruction syscall in a MIPS program.