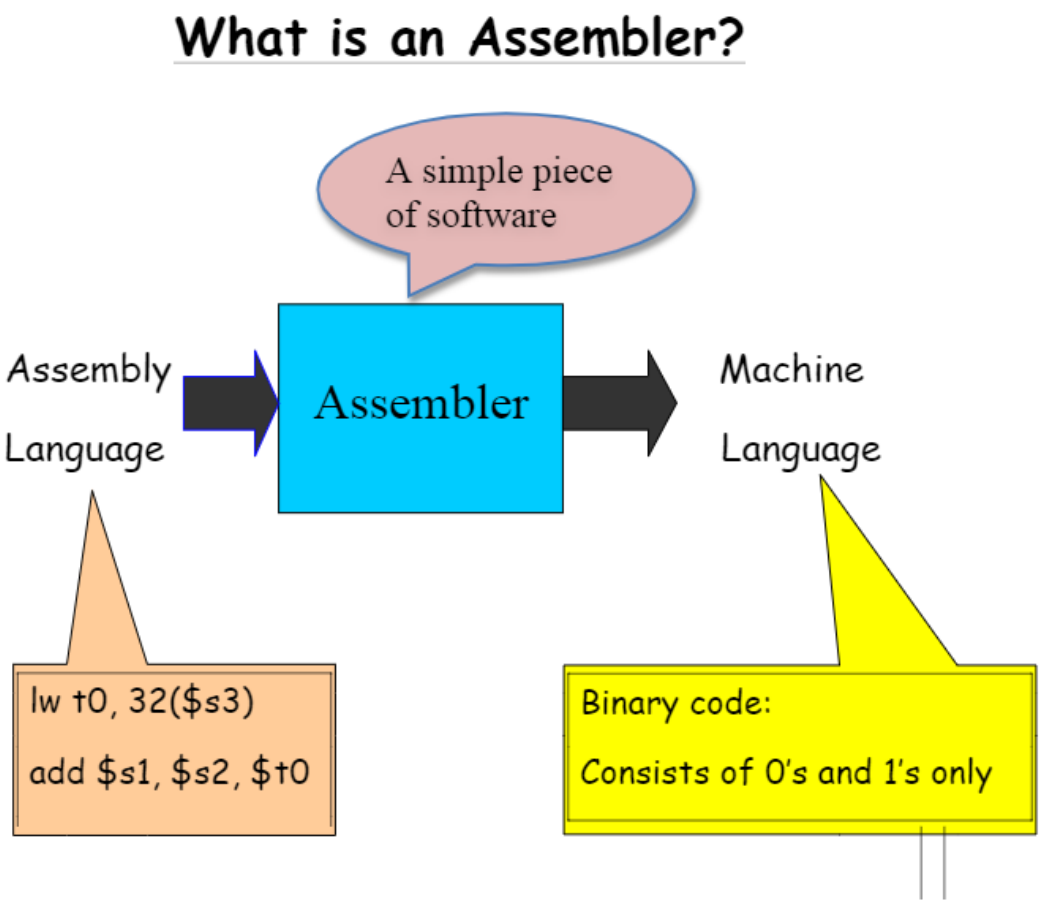
DHA Suffa University
Department of Computer Science
Computer Organization & Assembly Language
Spring 2021
Lab # 1(Introduction to MIPS & MARS Simulator)

What is Assembly Language?

An **assembly language** is a **programming language** that can be used to directly tell the computer what to do. An **assembly language** is almost exactly like the machine **code** that a computer can understand, except that it uses **words** in place of numbers



Objective

How to run the MARS simulator. A small MIPS program is used as an example.

Starting MARS

MARS, the **M**ips **A**ssembly and **R**untime **S**imulator, will assemble and simulate the execution of MIPS assembly language programs. It can be used either from a command line or through its integrated development environment (IDE). MARS is written in Java and requires at least Release 1.5 of the J2SE Java Runtime Environment (JRE) to work. It is distributed as an executable JAR file.

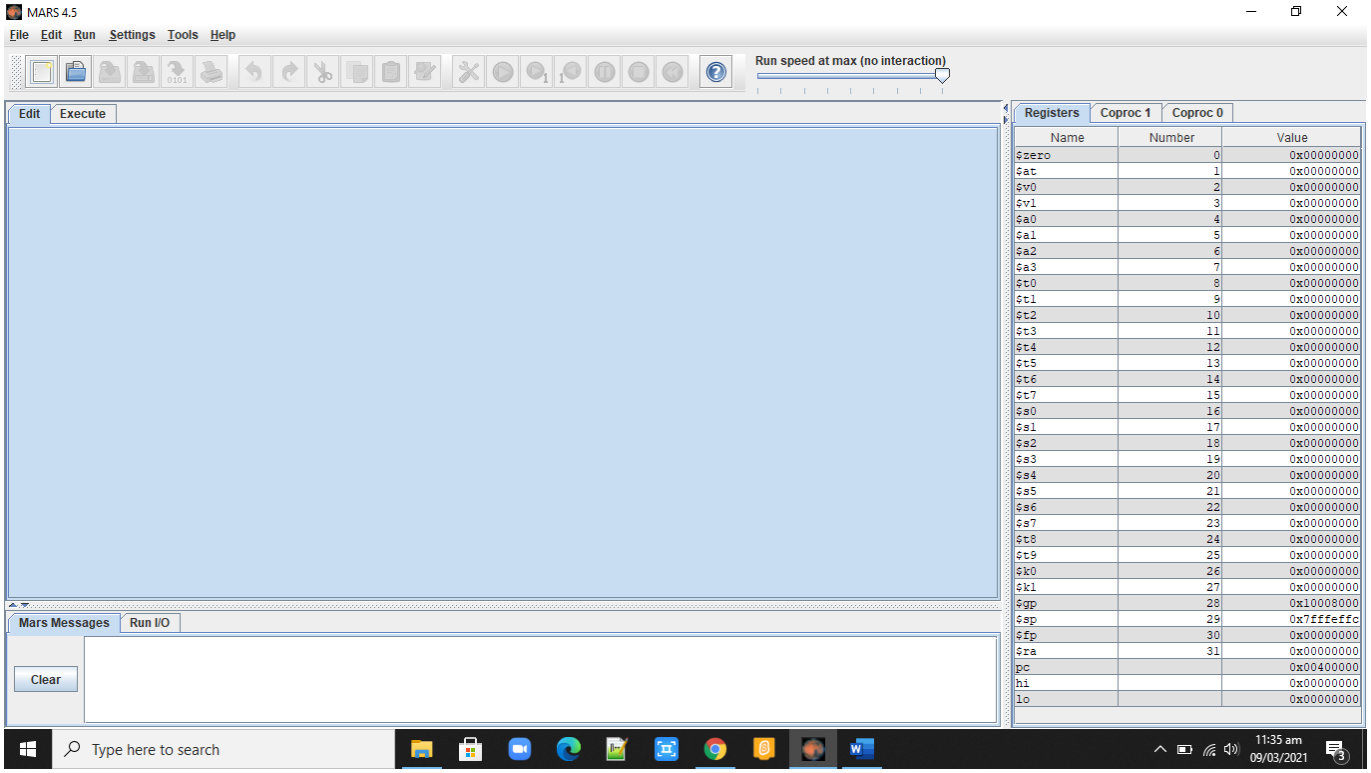
Understanding MARS Interface

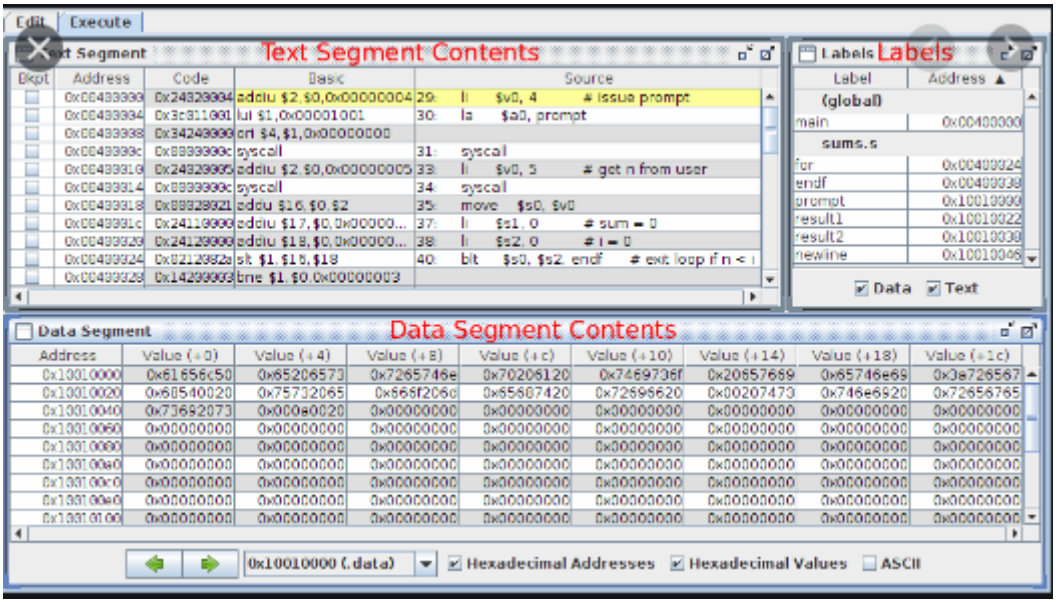
The IDE is invoked when MARS is run with no command arguments, e.g. java -jar mars.jar. It may also be launched from a graphical interface by double-clicking the mars.jar icon that represents this executable JAR file. The IDE provides basic editing, assembling and execution capabilities. Hopefully it is intuitive to use. Here are comments on some features.

- **Menus and Toolbar**: Most menu items have equivalent toolbar icons. If the function of a toolbar icon is not obvious, just hover the mouse over it and a tool tip will soon appear. Nearly all menu items also have keyboard shortcuts. Any menu item not appropriate in a given situation is disabled.
- **Editor**: MARS includes two integrated text editors. The default editor, new in Release 4.0, features syntax-aware color highlighting of most MIPS language elements and popup instruction guides. The original, generic, text editor without these features is still available and can be selected in the Editor Settings dialog. It supports a single font which can be modified in the Editor Settings dialog. The bottom border of either editor includes the cursor line and column position and there is a checkbox to display line numbers. They are displayed outside the editing area. If you use an external editor, MARS provides a convenience setting that will automatically assemble a file as soon as it is opened. See the Settings menu.
- **Message Areas**: There are two tabbed message areas at the bottom of the screen. The *Run I/O* tab is used at runtime for displaying console output and entering console input as program execution progresses. You have the option of entering console input into a pop-up dialog then echoes to the message area. The *MARS Messages* tab is used for other messages such as assembly or runtime errors and informational messages. You can click on assembly error messages to select the corresponding line of code in the editor.
- **MIPS Registers**: MIPS registers are displayed at all times, even when you are editing and not running a program. While writing your program, this serves as a useful reference for register names and their conventional uses (hover mouse over the register name to see tool tips). There are three register tabs: the Register File (integer registers $0 through $31 plus LO, HI and the Program Counter), selected Coprocesor 0 registers (exceptions and interrupts), and Coprocessor 1 floating point registers.
- **Assembly**: Select *Assemble* from the *Run* menu or the corresponding toolbar icon to assemble the file currently in the Edit tab. Prior to Release 3.1, only one file could be assembled and run at a time. Releases 3.1 and later provide a primitive Project capability. To use it, go to the *Settings* menu and check *Assemble operation applies to all files in current directory.* Subsequently, the assembler will assemble the current file as the "main" program and also assemble all other assembly files (*.asm; *.s) in the

same directory. The results are linked and if all these operations were successful the program can be executed. Labels that are declared global with the ".globl" directive may be referenced in any of the other files in the project. There is also a setting that permits automatic loading and assembly of a selected exception handler file. MARS uses the MIPS32 starting address for exception handlers: 0x80000180.

- **Execution**: Once a MIPS program successfully assembles, the registers are initialized and three windows in the Execute tab are filled: *Text Segment*, *Data Segment*, and *Program Labels*. The major execution-time features are described below.
- **Labels Window**: Display of the Labels window (symbol table) is controlled through the Settings menu. When displayed, you can click on any label or its associated address to center and highlight the contents of that address in the Text Segment window or Data Segment window as appropriate.





MIPS registers

register	assembly name	Comment
r0	$zero	Always 0
r1	$at	Reserved for assembler
r2-r3	$v0-$v1	Stores results
r4-r7	$a0-$a3	Stores arguments
r8-r15	$t0-$t7	Temporaries, not saved
r16-r23	$s0-$s7	Contents saved for use later
r24-r25	$t8-$t9	More temporaries, not saved
r26-r27	$k0-$k1	Reserved by operating system
r28	$gp	Global pointer
r29	$sp	Stack pointer
r30	$fp	Frame pointer
r31	$ra	Return address

Syscall Services

Service	$v0	Arguments / Result
Print Integer	1	$a0 = integer value to print
Print Float	2	$f12 = float value to print
Print Double	3	$f12 = double value to print
Print String	4	$a0 = address of null-terminated string
Read Integer	5	Return integer value in $v0
Read Float	6	Return float value in $f0
Read Double	7	Return double value in $f0
Read String	8	$a0 = address of input buffer $a1 = maximum number of characters to read
Allocate Heap memory	9	$a0 = number of bytes to allocate Return address of allocated memory in $v0
Exit Program	10	


```
# Program Adding 2 numbers
.data
msg: .asciiz "Sum of 2 Numbers is = \n"
.text
li $t0 , 4
li $t1 , 3

add $t0, $t1, $t0
li $v0 , 1
move $a0 , $t0
syscall
li $v0, 10
syscall

## End of file
```

The first “#” of the first line is in column one. The character “#” starts a comment; everything on the line from “#” to the right is ignored. Sometimes I use two in a row for emphasis, but only one is needed.

Basic Info About Running the Program

- Assemble of the program is necessary before running it
- This assembling loads the program in the OS.
- If there is an existing file so we should open it using open option in file tab and use it.
- If there are mistakes in your file, MARS’s message display panel shows the error messages. Use your editor to correct the mistakes, save the file then re-open the file in MARS.

Running a program:

Loading the source file into MARS does two things: (1) The file is assembled into machine instructions, and (2) the instructions are loaded into MARS’s memory. The text display shows the result.

The text display is the second window from the top. You should see some of the source file in it and the machine instructions they assembled into. The leftmost column are addresses in simulated memory.

Explanation of the Program:

There are various ways for a program executing on a real machine to return control to the operating system. But we have no OS, so for now we will single step instructions. Hopefully you are wondering how the program works.

The first line of the program is a comment. It is ignored by the assembler and results in no machine instructions.

.text is a directive. A directive is a statement that tells the assembler something about what the programmer wants, but does not itself result in any machine instructions. This directive tells the assembler that the following lines are “text” -- source code for the program.

Blank lines are ignored. The line main: defines a symbolic address (sometimes called a statement label). A symbolic address is a symbol (an identifier) that is the source code name for a location in memory. In this program, main stands for the address of the first machine instruction (which turns out to be 0x00400000). Using a symbolic address is much easier than using a numerical address. With a symbolic address, the programmer refers to memory locations by name and lets the assembler figure out the numerical address.

The symbol main is global. This means that several source files can use the symbol main to refer to the same location in storage. (However, SPIM does not use this feature. All our programs will be contained in a single source file.)

Assembly Language Statement

The layout of a machine instruction is part of the architecture of a processor chip. Without knowing the layout, you can't tell what the instruction means. Even if you know the layout, it is hard to remember what the patterns mean and hard to write machine instructions.

A statement in pure assembly language corresponds to one machine instruction. Assembly language is much easier to write than machine language. Here is the previous machine instruction and the assembly language that it corresponds to:

machine instruction	**assembly language statement**
0000 0001 0010 1011 1000 0000 0010 0000	**add $t0, $t1, $t2**

The instruction means: add the integers in registers $t1 and $t2 and put the result in register $t0. To create the machine instruction from the assembly language statement a translation program called an assembler is used.

Humans find assembly language much easier to use than machine language for many reasons.

- It is hard for humans to keep track of those ones and zeros.
- By using symbols programmers gain flexibility in describing the computation.

LAB TASK 01

(i) Write a program in MIPS to solve following expression:

(9-8)*12+(16-15)-(5*2)

LAB ASSIGNMENT 01

Write a program in MIPS to solve following expressions:

(i) ((52*8)*12+8)-(18+116)+(15*2)

(ii) ((32+52)+58*3)+57