#6 : MIPS Programming II

Computer Architecture 2021/2022

Ricardo Rocha

Computer Science Department, Faculty of Sciences, University of Porto

MARS – MIPS Simulator

Main functionalities:

- Edit programs (**assembly**)
- Compile (**assembler**)
- Run and/or execute step by step
- See the memory contents and the values in the set of registers

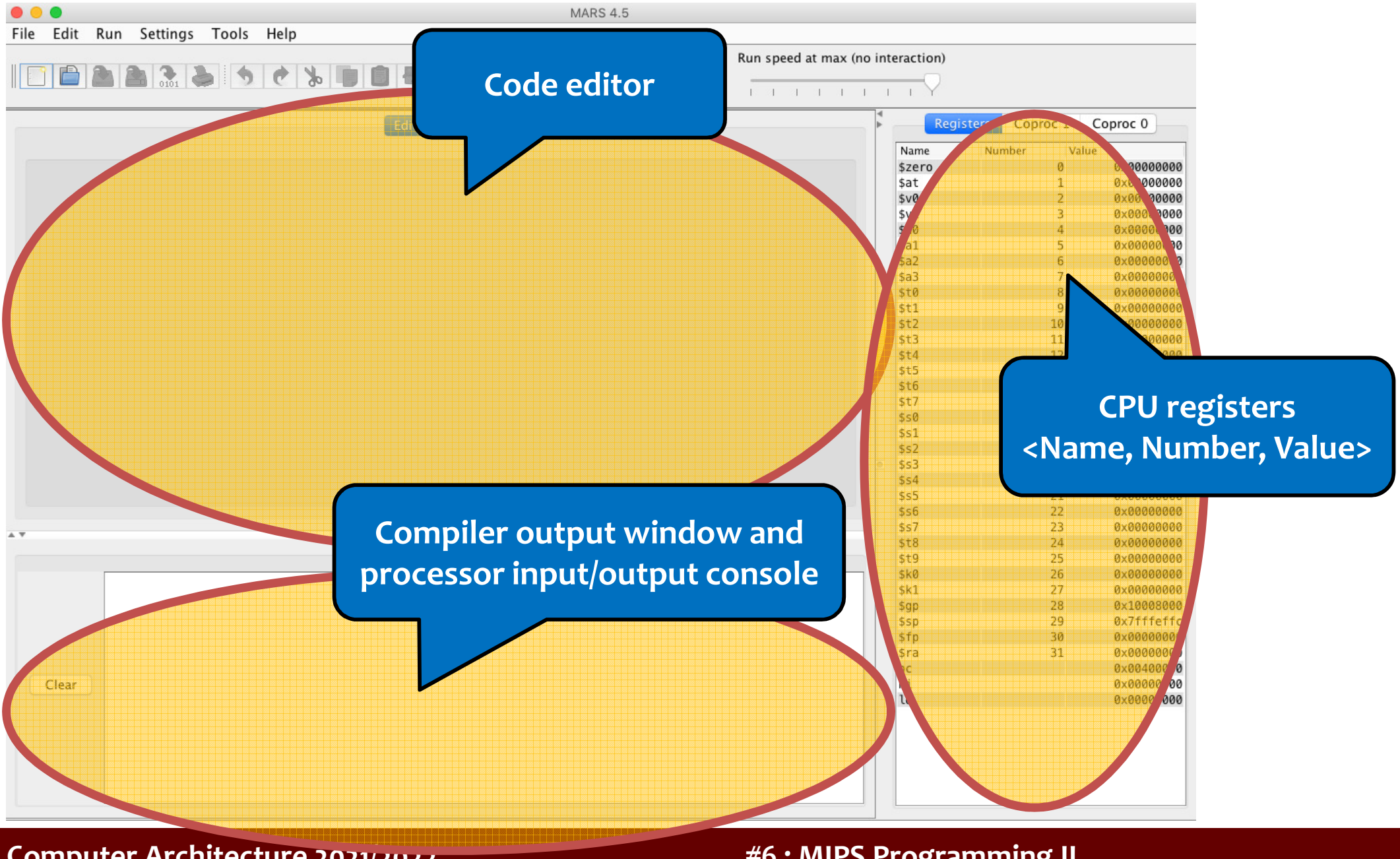
Download Mars4_5.jar:

- http://www.softpedia.com

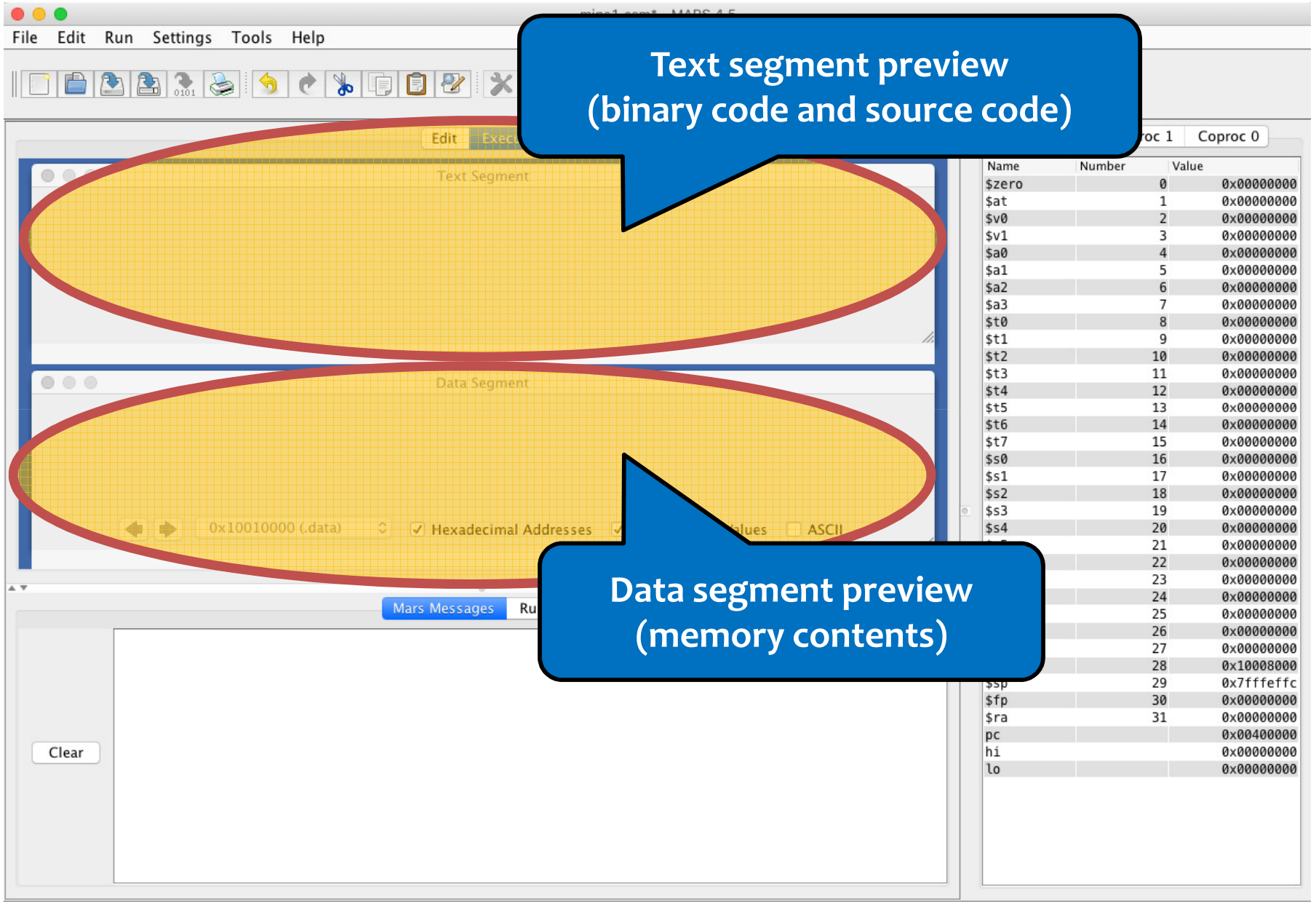
Command to execute:

- java –jar Mars4_5.jar

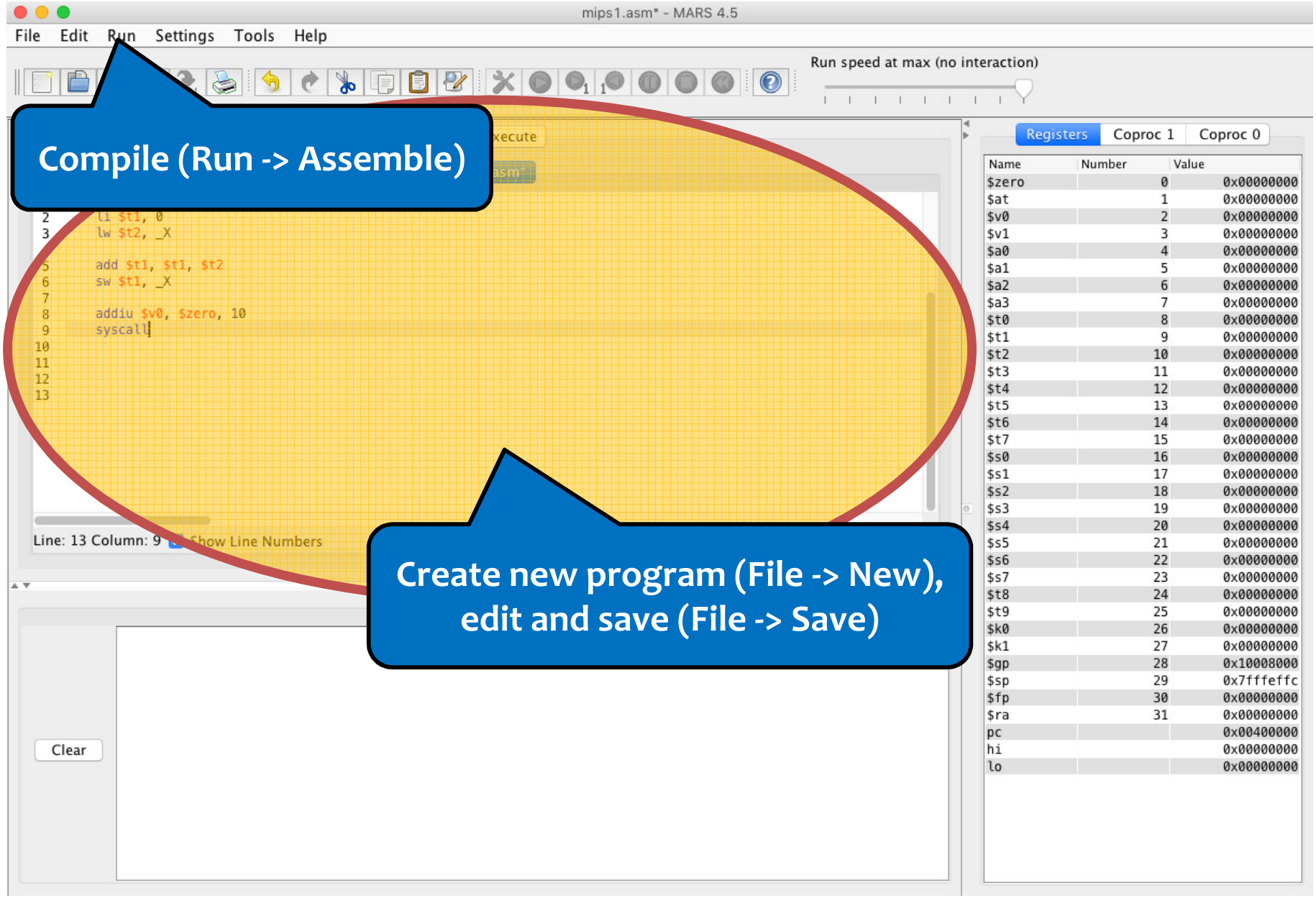
MARS – MIPS Simulator



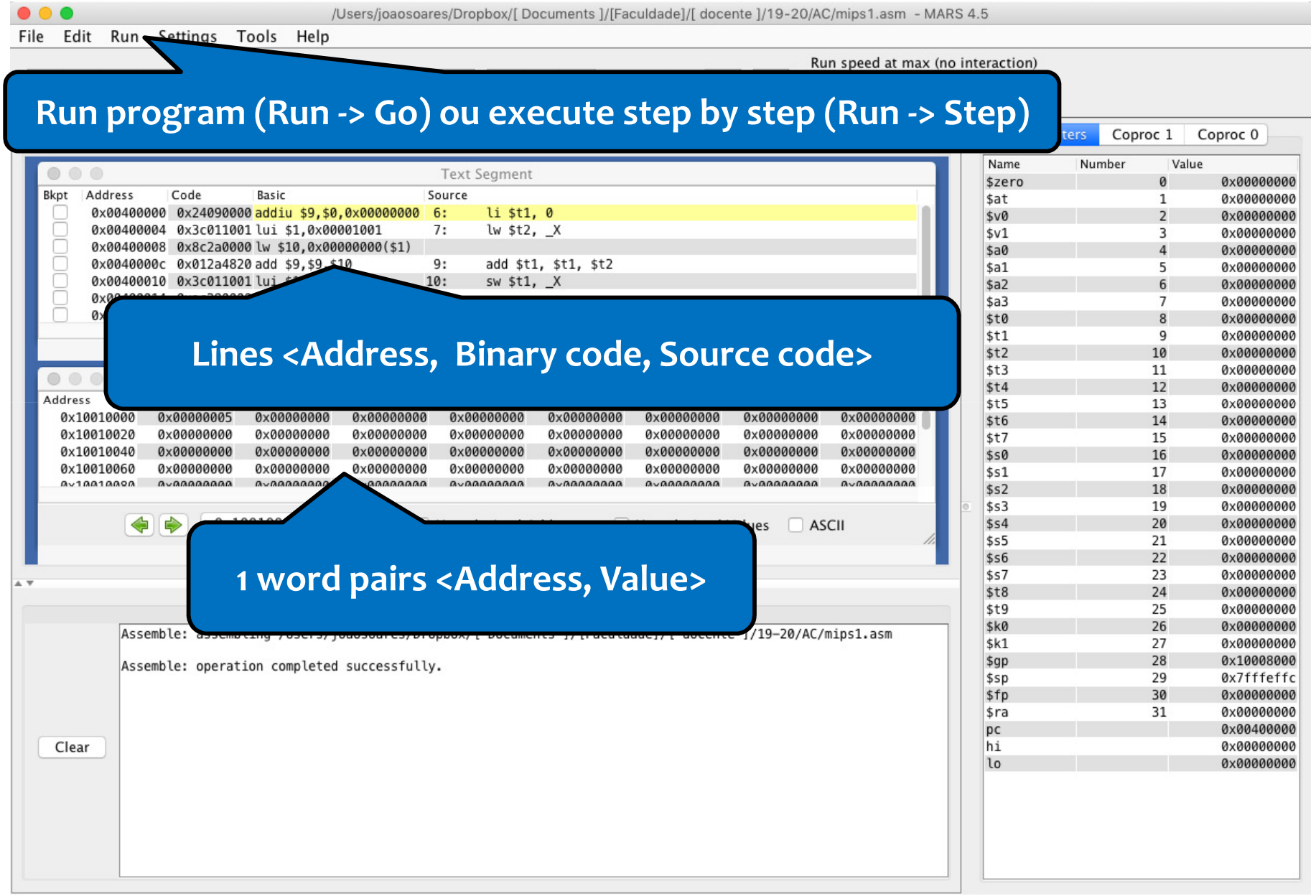
MARS – MIPS Simulator



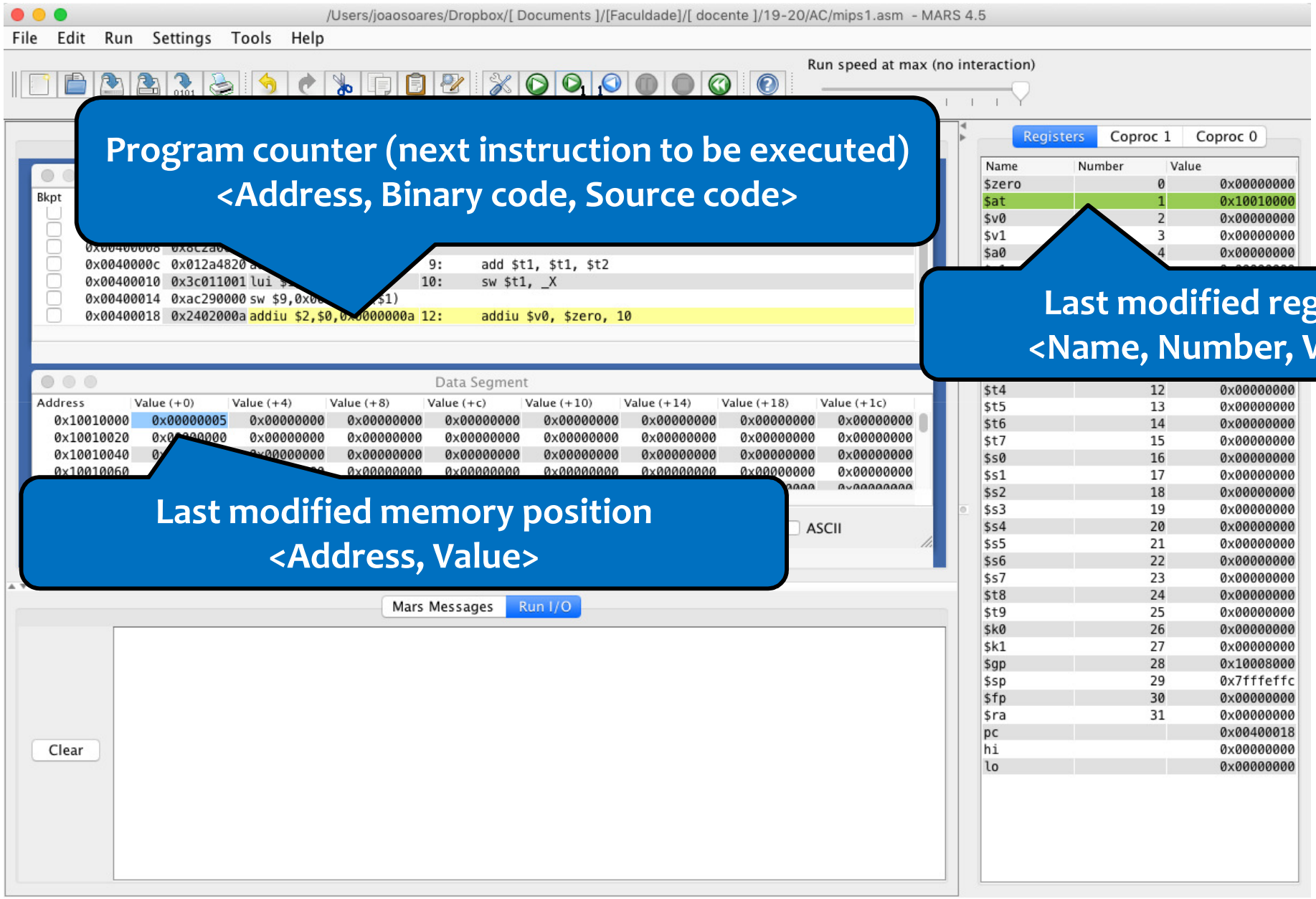
MARS – MIPS Simulator



MARS – MIPS Simulator



MARS – MIPS Simulator



Procedure Calls

The execution of a procedure call happens when one procedure (the **caller**) invokes another procedure (the **callee**). In general, the execution of a procedure call follows six steps:

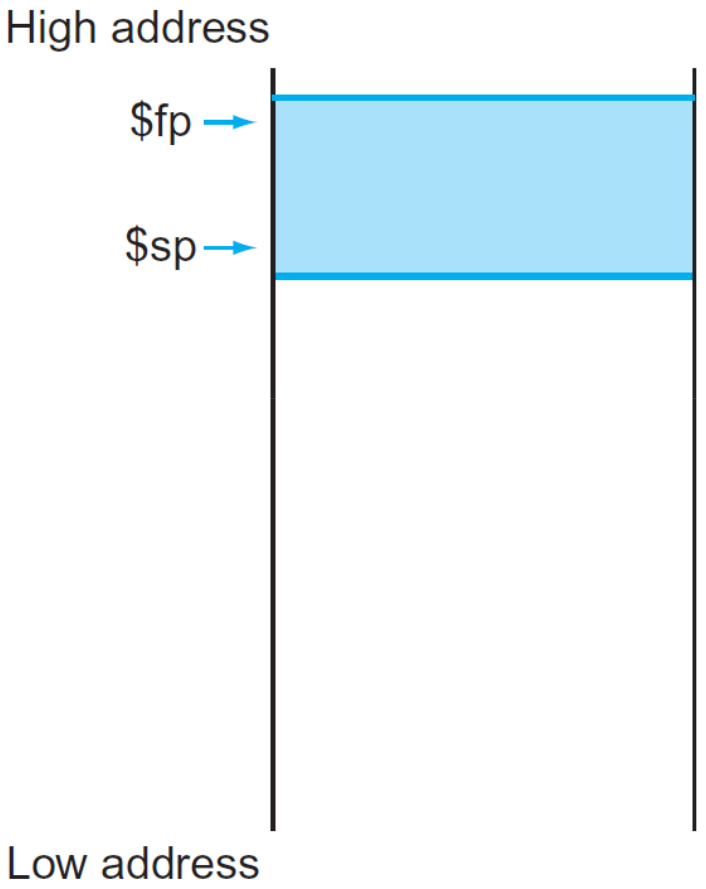
- Put arguments in a place where the callee can access them
- Transfer control to the callee
- Acquire storage resources needed for callee execution
- Perform callee's operations
- Put results in a place where the caller can access them
- Return control to the caller's next instruction

Procedure Calls

The bookkeeping associated with procedure calls is done in the **stack segment** around blocks of memory called **procedure frames**.

By historical precedent, the **stack grows from higher addresses to lower addresses**. This convention means that you push values onto the stack by subtracting from the stack pointer.

- Register **$fp (frame pointer)** points to the **base of the current procedure frame** and offers a stable base register as it does not change in a procedure
- Register **$sp (stack pointer)** points to the **top of the current procedure frame** and can change within a procedure



Preserved or Not Preserved

What is preserved across a procedure call?

- **$sp is preserved** by the callee by adding exactly the same amount that was subtracted from it
- **Stack above $sp is preserved** by making sure the callee does not write above $sp, i.e., the caller will get the same data back on a load from the stack as it was stored there
- **Other registers can be preserved** by saving them on the stack (if they are used) and restoring them from there, specially **registers $s0–$s7** and **register $ra**

Preserved	Not preserved
Saved registers: `$s0–$s7`	Temporary registers: `$t0–$t9`
Stack pointer register: `$sp`	Argument registers: `$a0–$a3`
Return address register: `$ra`	Return value registers: `$v0–$v1`
Stack above the stack pointer	Stack below the stack pointer

Procedure Call Support

MIPS conventions for procedure calling:

- **$a0 – $a3 registers** are used to **pass the first 4 arguments to the callee**
- **$v0 – $v1 registers** are used to **return values to the caller**
- **$t0 – $t9 registers** are used to **hold temporary values** that can be overwritten by the callee
- **$s0 – $s7 registers** are used to **hold long-lived values** that should be preserved across calls
- **$sp register** is the pointer to the **current top location in the stack**
- **$ra register** is the return address to the **caller's next instruction**
- **jump-and-link instruction (jal)** jumps to an address and simultaneously saves the address of the following instruction **(PC + 4)** in register **$ra**
- **jump register instruction (jr)** jumps to the address stored in register **$ra**

Caller Side

Save not preserved registers

- If the caller expects to use not preserved registers ($t0 – $t9, $a0 – $a3 and $v0 – $v1) after the call, save its values before the call in the current procedure frame

Pass arguments

- The first 4 arguments are put in registers $a0 – $a3
- Additional arguments are pushed on the stack and appear at the beginning of the procedure frame (register $fp points to the base of the procedure frame)

Transfer control to the callee

- Execute a jal instruction to jump to the callee's first instruction and save the return address in $ra

Callee Side

Allocate memory (and update stack pointer)

- Add a new procedure frame by subtracting the required size from $sp

Save preserved registers (and update frame pointer)

- If the callee expects to alter preserved registers ($fp, $ra and $s0 – $s7), save its values in the new procedure frame before altering them ($fp only needs to be saved if the frame's size is not zero; $ra only needs to be saved if the callee itself makes a call)
- Update $fp by adding the new frame's size minus 4 to $sp

Put results and return control to the caller

- If the callee returns something, put the result(s) in $v0 – $v1
- Restore all callee-saved registers ($fp, $ra and $s0 – $s7)
- Pop the procedure frame by adding its size to $sp
- Execute a jr instruction to return by jumping to the address in $ra

Simple Procedure Call

```
int proc (int arg1, int arg2) {   // arguments in $a0 and $a1
   int r = ...;                   // r in $s0, need to save $s0 on stack
   return r;                      // return value in $v0
}
-------------------------------------------------------------------
_main: ...
       li    $a0, ...             # put argument $a0
       li    $a1, ...             # put argument $a1
       jal   _proc                # jump and link
       ...

_proc: addiu $sp, $sp, -4         # adjust stack pointer
       sw    $s0, 0($sp)          # save $s0
       ...                        # return value in $v0
       lw    $s0, 0($sp)          # restore $s0
       addiu $sp, $sp, 4          # restore stack pointer
       jr    $ra                  # return
```