COMP2611: Computer Organization

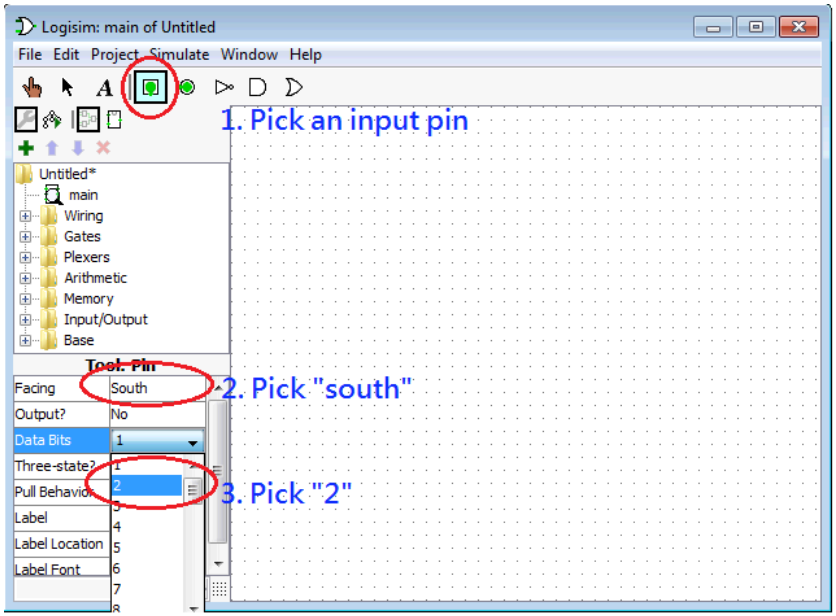
Building a 2-bit register and introduction to MARS

Building a 2-bit register with the D Flip-Flops

- Continue from the last lab.
- We can build a simple register that holds 2 bits with the D flip-flop (you can use the D flip-flop to build register of any size, we build a 2-bit register here for simplicity).
- Basically, the idea is to use two D flip-flops in the circuit, each of which holds a single bit.
- The input will be 2-bit in width. We need to separate the individual bits by using a "splitter" and direct the bits to the corresponding D flip-flops.

Building a 2-bit register with the D Flip-Flops

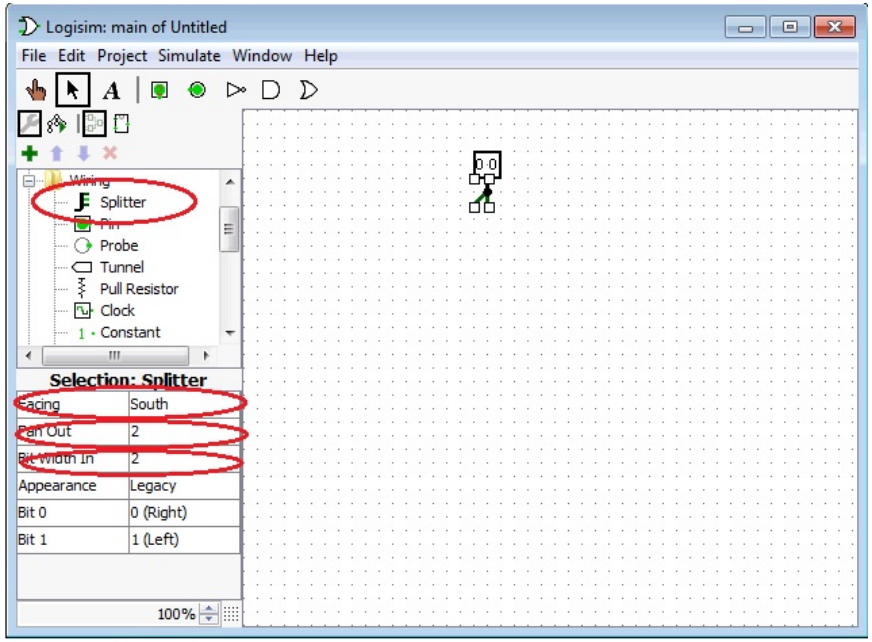
- To have 2-bit inputs, you need to follow the 3 steps below to set up the input pin.



- When you pick "south" for "Facing" in step 2, wires can be drawn from the bottom (south) of the input pin. It is up to you whether you want to choose "south".
- After the 3 steps you drag the pin to the canvas and you will have this 2-bit input pin on the canvas (values have been initialized to 0,0).

Building a 2-bit register with the D Flip-Flops

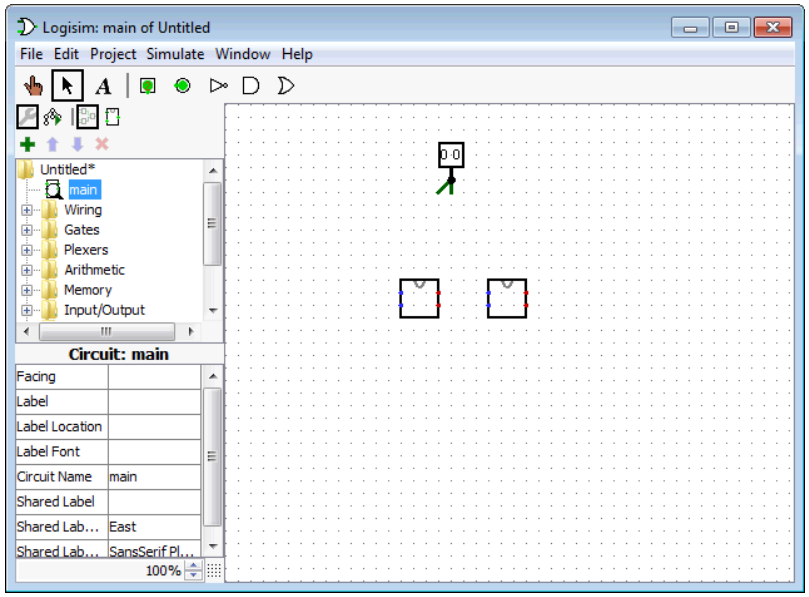
- Select a 2-bit splitter from the "wiring" folder of the "Explore pane".



- Make sure you have set the "Fan-Out" (number of output legs) and "Bit Width In" both to be 2. So that each leg carries 1 bit of data.

Building a 2-bit register with the D Flip-Flops

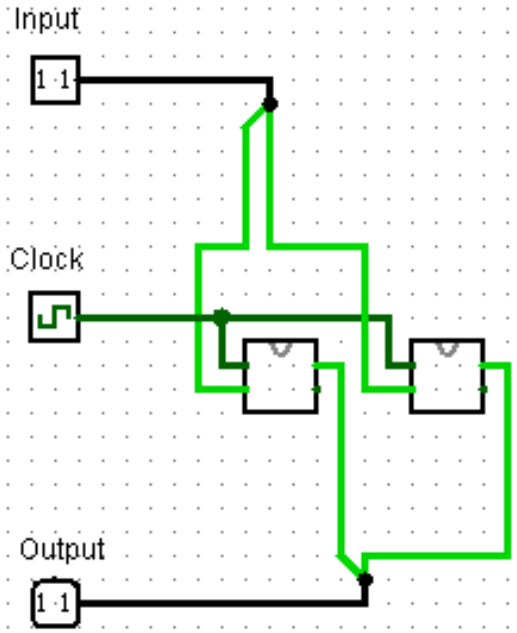
- Now load the "D-flip-flop-withoutclock.circ" as a library and use it as a component of the register. You can load it by clicking "Project", "Load Library", "Logisim library…" and select "D-flip-flop-withoutclock.circ".
- Drag two D flip-flops from the "Explorer pane".



- Note the two input dots (in blue) on the left of the flip-flop and the two output dots (in red) on the right.
- The upper input dot is C (Clock), the lower input dot is D (Data).
- The upper output dot is Q and the lower output dot is ~Q.

Building a 2-bit register with the D Flip-Flops

- To store 2 bits into the flip-flops, you need to forward the bits from the splitter to the corresponding D flip-flops.
- You also need to connect a clock signal to both of the D flip-flops.
- Don't forget to retrieve the output from the flip-flops (using an output pin).
- Here is an example circuit for the register:



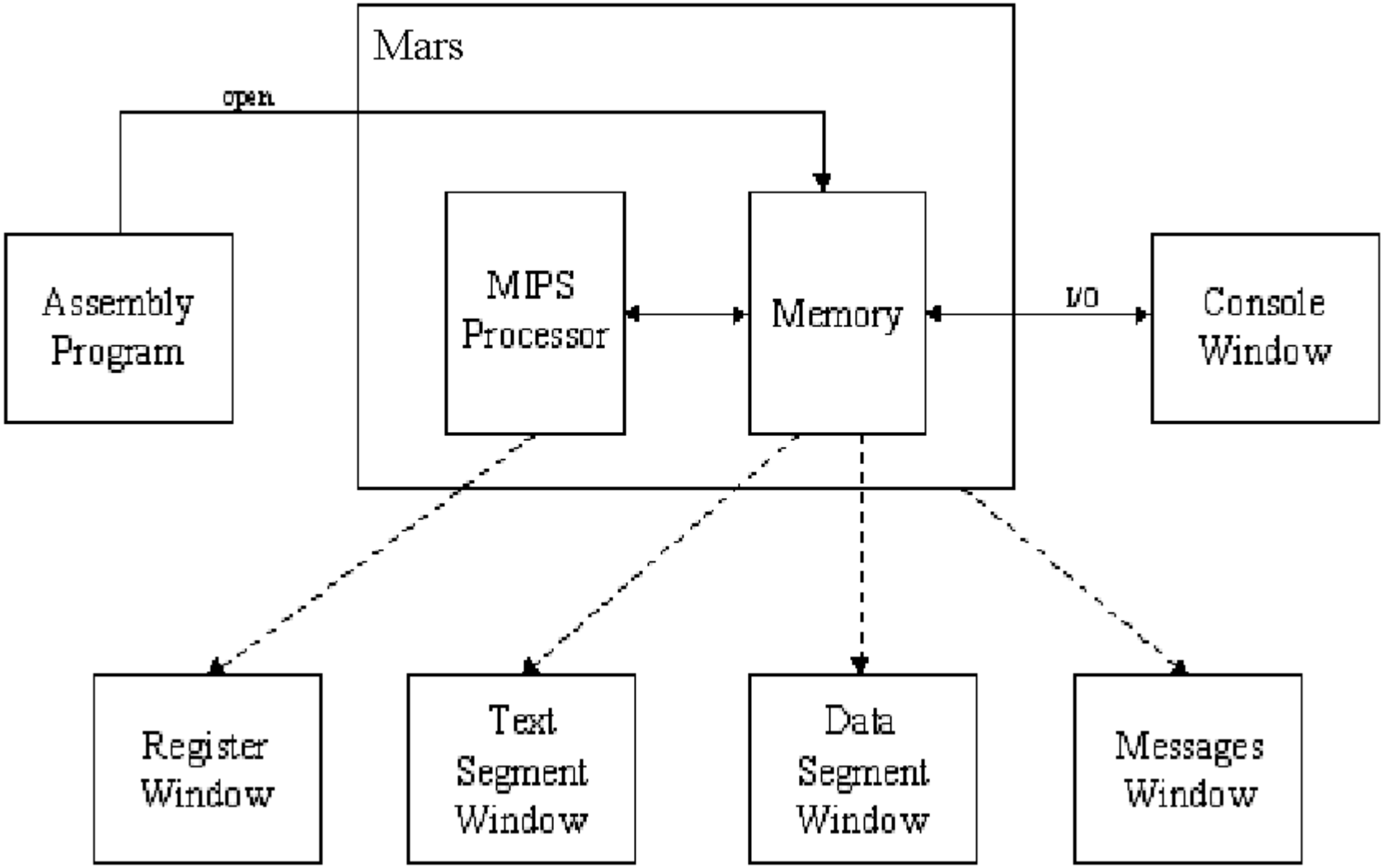
Building a 2-bit register with the D Flip-Flops

- Some hints:
  - The colors of the wires hint you on possible errors, make good use of this information.
  - Each of the input/output pins can be connected in one direction only, as determined by the "facing" attribute. Make sure you connect it correctly.
  - A splitter splits bits according to their location in the input, the order is preserve on the splitter, (i.e., left-most bit on the left-most split wire and so on)
  - Logisim could sometimes have mysterious errors on the wires even though there is no error at all. An easy way to solve this is to copy and paste the whole circuit to a new canvas. This usually solves the problem.

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

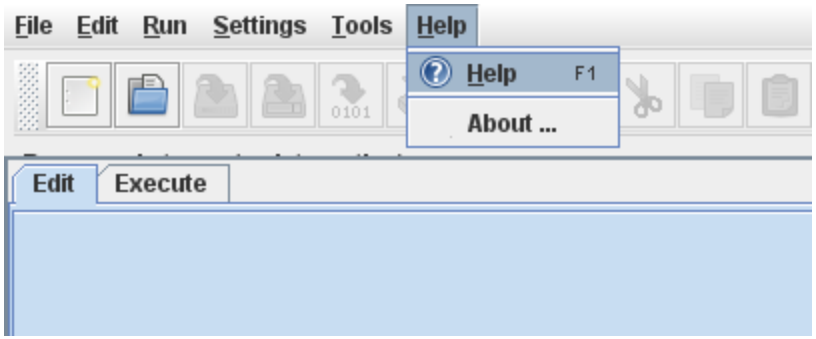


Getting and installing JRE

- 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.

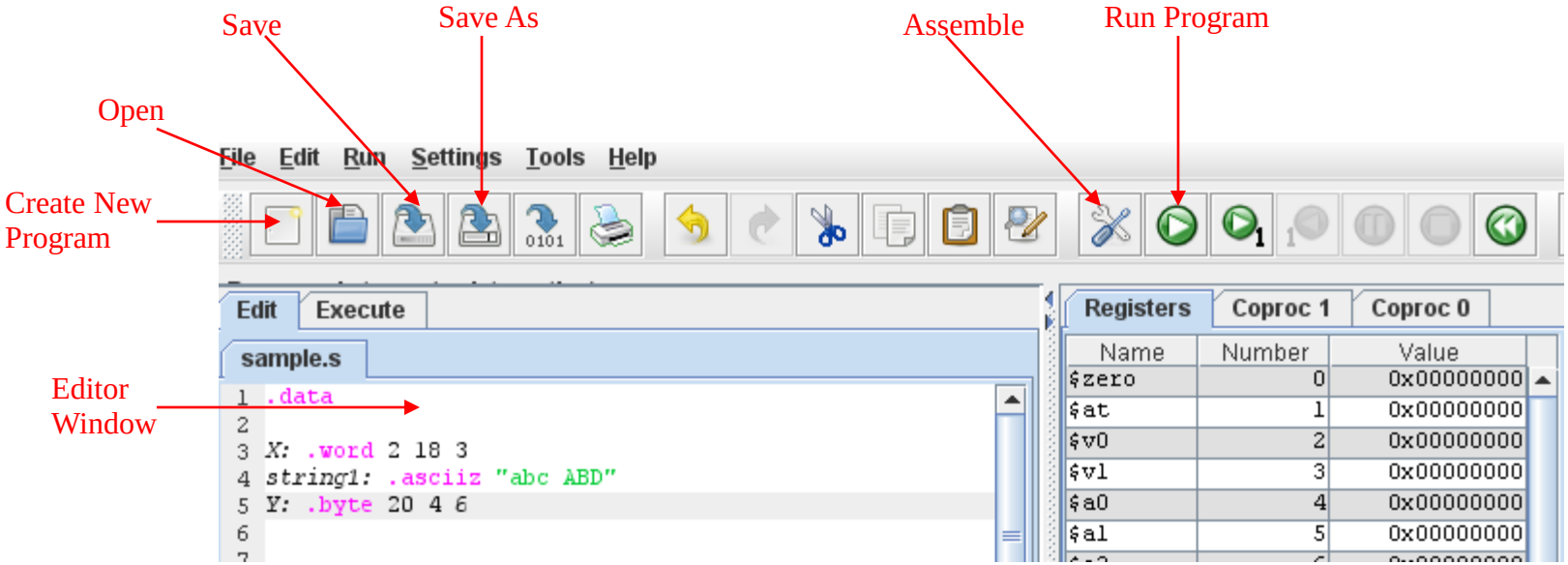
Getting and running MARS

- 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 https://course.cse.ust.hk/comp2611/MARS_4_5.jar, 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.



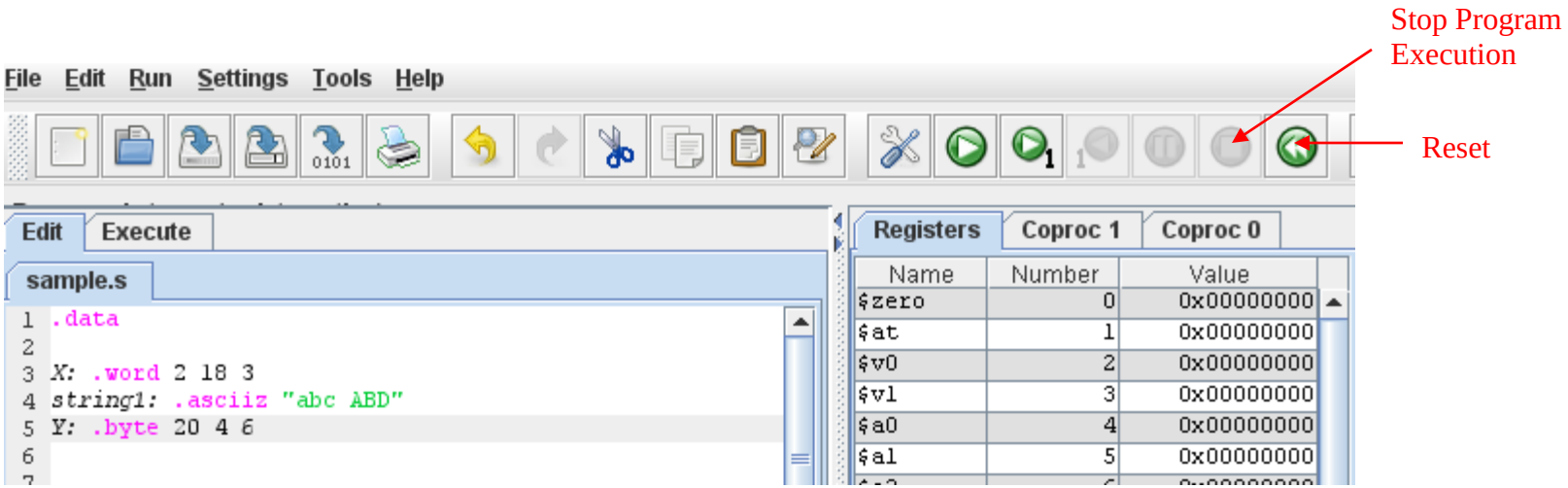
Running an assembly program

- 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.

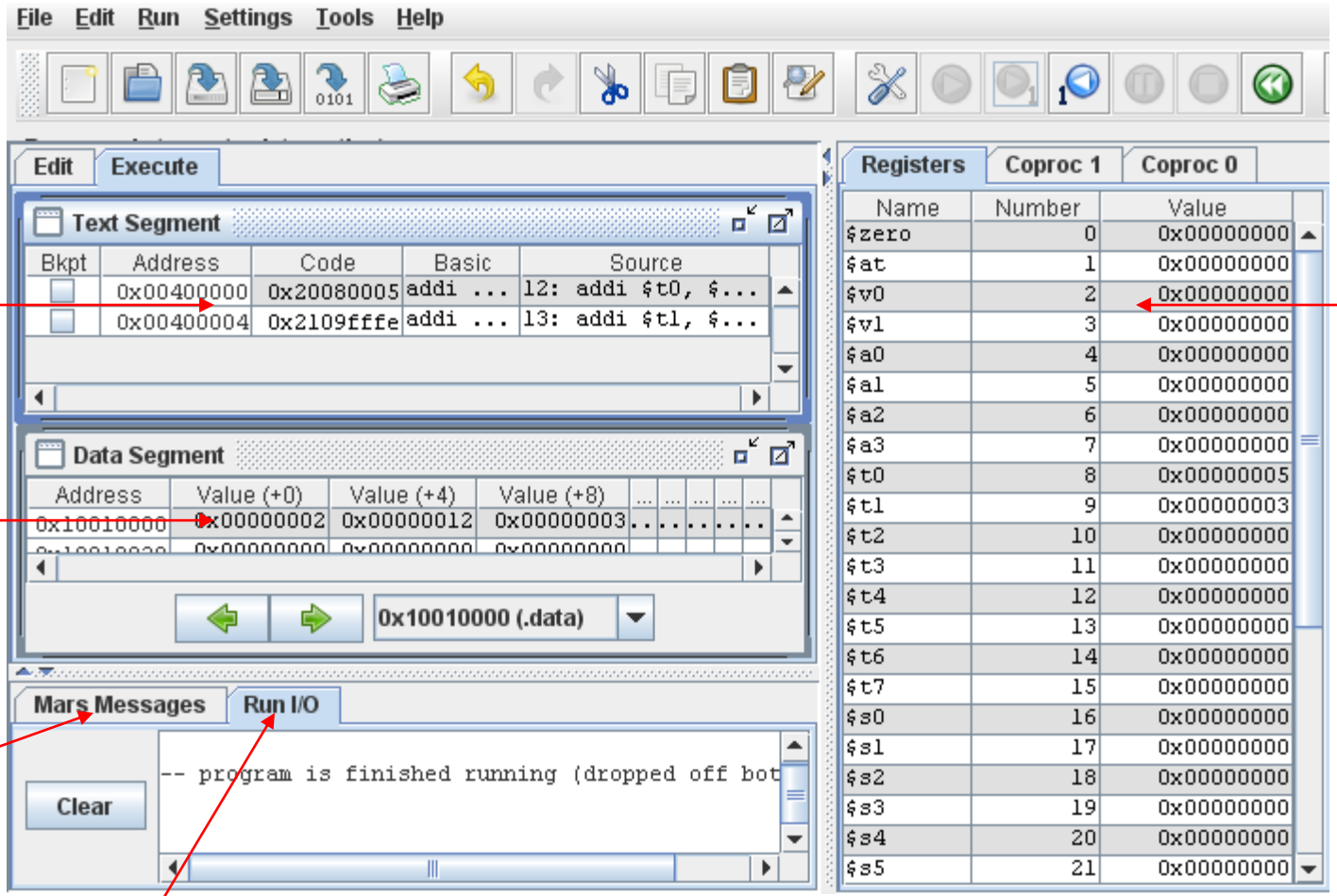


Example program

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.

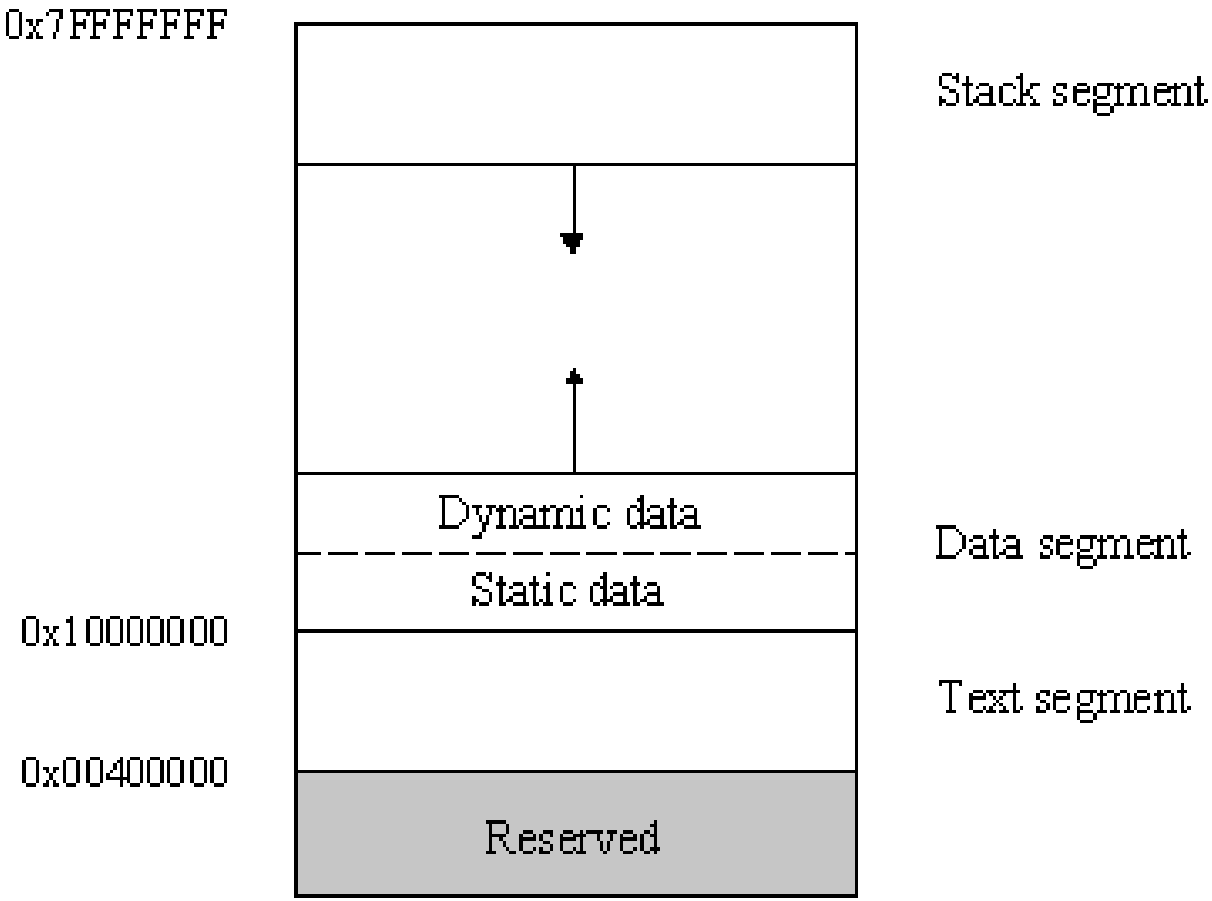
Registers Window

- 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



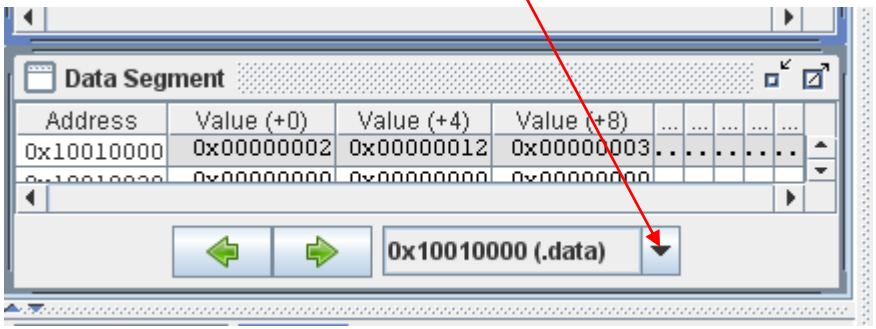
Text Segment Window

- **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.

Conclusion

- You have:
  - finished building the 2-bit register,
  - 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.