SCC150 MIPS/Assembly Week 11 Practical

Haris Rotsos and Angie Chandler

SCC – Lancaster

Week 11

Outline

- MARS MIPS Simulator
- Using add
- Register initialisation
- Bit manipulation
- Memory manipulation

MARS Introduction

- MARS is a MIPS simulator written in Java
  - Simulates a MIPS CPU
  - Executes your MIPS program
- Using MARS you can:
  - Enter MIPS assembly commands
  - Monitor/edit registers
  - Monitor/edit memory
  - Debug MIPS code

Remark
This is only a MIPS simulator and does not reflect the Intel instructions and registers that run on the processor of your workstation or laptop compute

MARS Installation

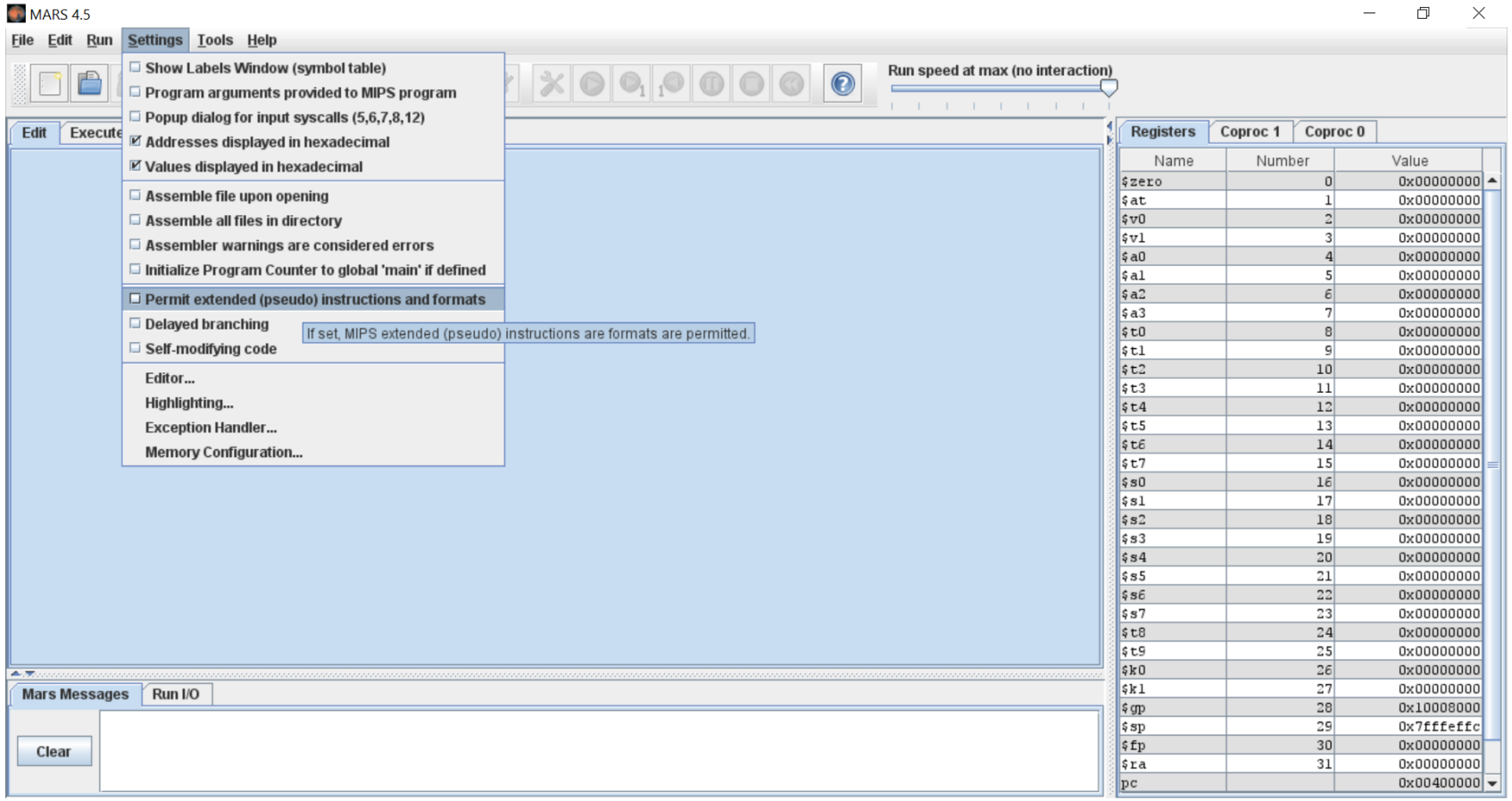
Note

MARS is a Java application and will run on any system that provides a Java runtime environment. If you have Java set up for 110, you don't need to install the Java runtime environment.

- Download JAVA JRE – https://www.java.com/eu/download/
- Download MARS – http://courses.missouristate.edu/KenVollmar/MARS/download.htm
- Double click the downloaded JAR file.
- Working on lab machines
  - MARS is pre-installed on all lab machines. A link is available on the start menu

Disable Pseudo Instructions

- Before you start, disable pseudo instruction support



Working on MIPS – in Pairs

This week provides you with an exploratory introduction to MIPS. As a result, we suggest that it might be best carried out with a partner, so that you can discuss result and explore further as you go along.

Find a partner to work with. If you don't know anyone to ask in your lab class, then ask the TA to help.

You should still both try all of the exercises, so that you both definitely understand what is going on, but take the opportunity to discuss why things work the way they do.

Exercise 1 – Using Add

- Task – use a single **add** instructions to perform each of the following computations
- Before stepping through your instruction, be sure to manually set the values of the source registers
- You can use any registers that you like; conventional ones to use are the temporary ($t) or saved ($s) registers
  - 5 + 5
  - 5 – 1
  - 2147483647 + 1 (note $2147483647 = (2^{31} - 1)$)
- Does the output match what you expect?

Exercise 2 – Register Initialisation

- You can use the add immediate (addi) instruction to specify a constant as one operand
- A MIPS register called $zero always contains the value zero.
- Task: write a sequence of three instructions to:
  - Set $t1 to hold a value of 8
  - Set $t2 to hold a value of 12
  - Compute the sum, storing the result in $t0

Exercise 2 – Large Constant Values

- What is the biggest constant value that you can set to a register, using addi? (Note: try different constant values for an addi instruction, eg powers of 2, 256 – 1024).
- lui register, const value (load upper immediate), sets the upper 4 bits of the register to the const value

```
lui $t1, 0xFFFF     #$t1 = 0xFFFF0000
```

- Task: Write a sequence of instructions to:
  - Set $t1 to hold a value of 2^{29}
  - Set $t2 to hold a value of ($2^{29}$ + 40)
  - Compute and store the sum in $t0

Exercise 3 – Using sll

- Store a value of 5 in a register
- Task: compute the result of the following operations:
  - Shifting left by one bit
  - Shifting left by two bits
  - Shifting left by three bits
- Based on the results, can you think of a possible application of sll in numeric calculation? Why does this work?

Memory Management in MIPS

- Programs (ie instructions) and data are both held in separate memory addressable regions, also called segments
- The memory area between 0x00400000 and 0x0ffffffc is the text segment and contains program instructions
- The data segment starts at 0x10000000
- To initialise $s0 as a pointer to the beginning of the data segment

```
lui $s0, 0x1000          #load address 0x10000000 in $s0
```

Exercise 4 - Memory Access

- Task: Write an assembly program that:
  - Writes the value 0101010101010101_2 in address 0x10000000. First put the value in a register and then store the register content in memory (Note: convert the 16-bit binary value into either decimal or hex first)
  - Write the value $10101010101010101010101010101010_2$ in address 0x10000004. (Hint: since constant values are 16-bits, load the upper 16 bits using lui, and the low 16 bits using ori).
  - Add the values stored at addresses 0x10000000 and 0x10000004, and store the result in address 0x10000008
- Does the result match your expectations?

Memory Monitoring

Debugging the data segment

To watch what's going on at that address in memory, you'll need to select 0x10000000 (.extern) within the data segment window of the MARS "Execute" tab.