MARS (MIPS Assembler/Simulator) Tutorial

1. Download MARS

1.1) Download MARS from:
http://courses.missouristate.edu/KenVollmar/MARS/download.htm .

1.2) Start the MARS simulator by double clicking on the .jar file. *MARS requires Java J2SE 1.4.2 (or later) SDK installed on your computer (http://java.sun.com/javase/downloads/index.jsp)*

1.3) MARS is a full featured MIPS assembly IDE, with a built-in editor where you can enter your assembly programs and assemble them along with a simulator that will run your MIPS assembly programs and allow you to debug them.

2. Input the Tutorial program

2.1) Open the MARS program and click from the file menu choose "File…New". A black document will open which you can enter your assembly code into. Click "File…Save As" and save the file as "tutorial01.asm".

2.2) Enter the code as shown below into the editor and save the file.

```
#-------------------------------------------------------------
# Program File: tutorial01.asm
# Written by:   Nate Houk
# Date Created: 1/22/08
# Description:  Tutorial program to introduce MARS simulator
#                       including: breakpoints, single-stepping,
#                       and register and memory windows.
#-------------------------------------------------------------

#-----------------------
# Declare some constants
#-----------------------
              .data
string1:      .asciiz       "Welcome to EE 109\n"
string2:      .asciiz       "Assembly language is fun!\n"
string3:      .asciiz       "\nLoop #"


#------------------
# Main program body
#------------------
.text
main:
li     $v0,   4
la     $a0,   string1
syscall
la     $a0,   string2
syscall
li     $t0,   1
loop:
li     $v0,   4
la     $a0,   string3
syscall
li     $v0,   1
move   $a0,   $t0
syscall
addi   $t0,   $t0,   1
bne    $t0,   4,            loop
#-----
# Halt
#-----
li  $v0, 10
syscall
```

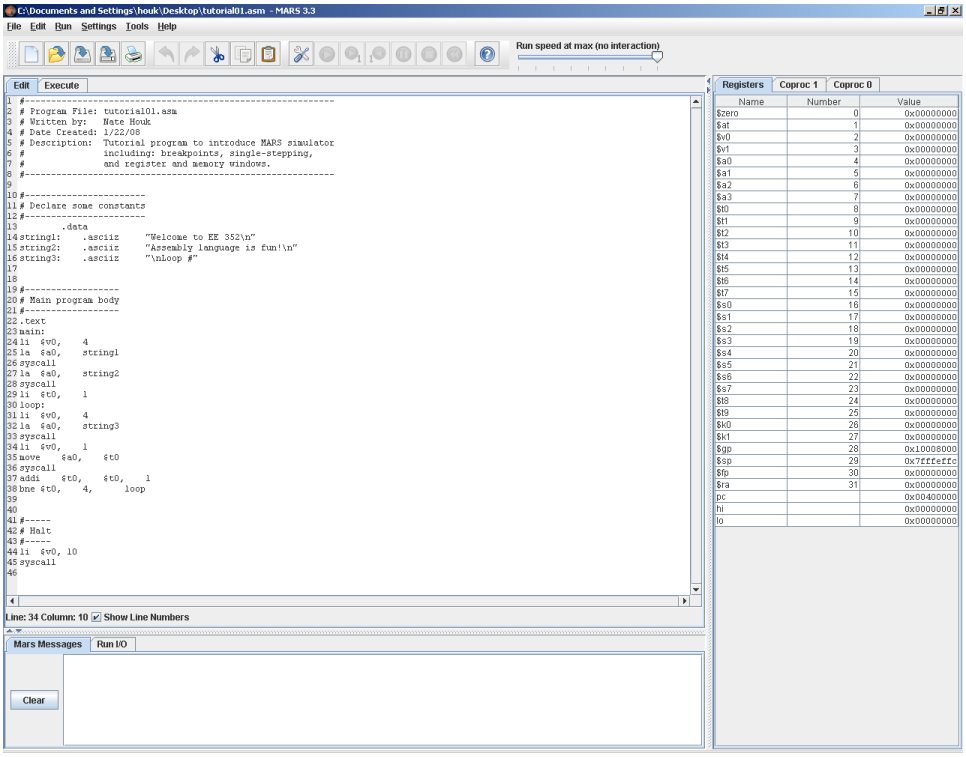


Figure 1 – MARS Editor

2.3) From the menu, choose "Run…Assemble". The "Mars Messages" window at the bottom of the screen will indicate if any errors occurred. No errors should occur.

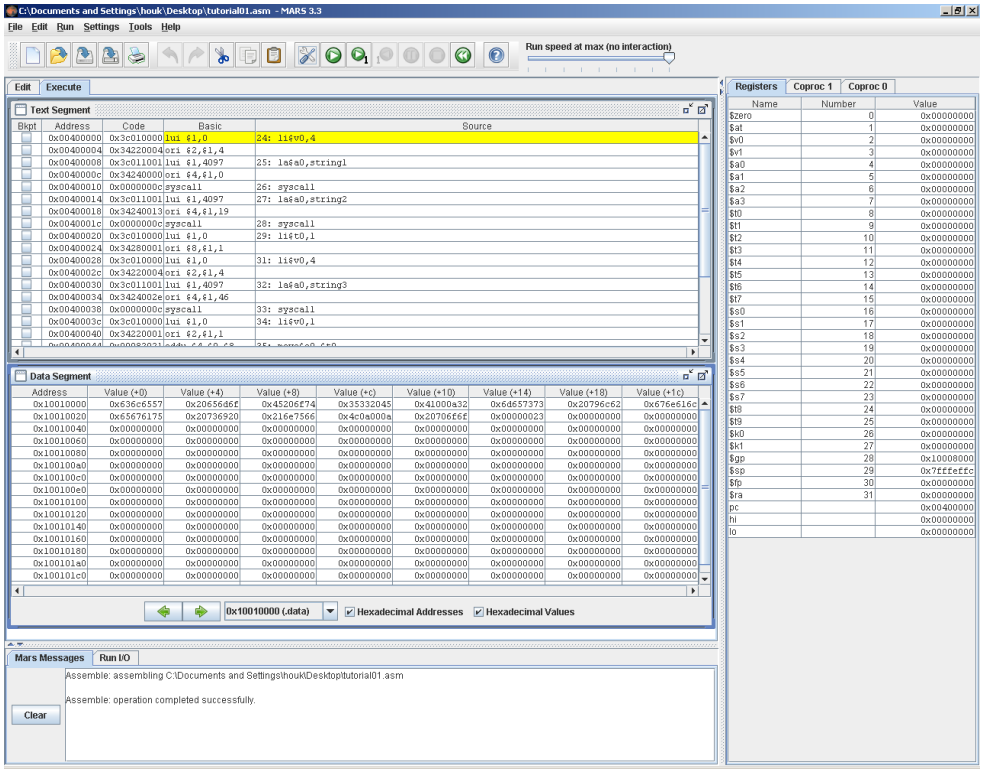


Figure 2 – MARS Simulator after Successful Assembly

3. Simulate the tutorial program

3.1) From the menu, choose “Run…Go” to execute the program. The program will execute displaying two lines of text and three iterations of a loop to the Run /IO window.

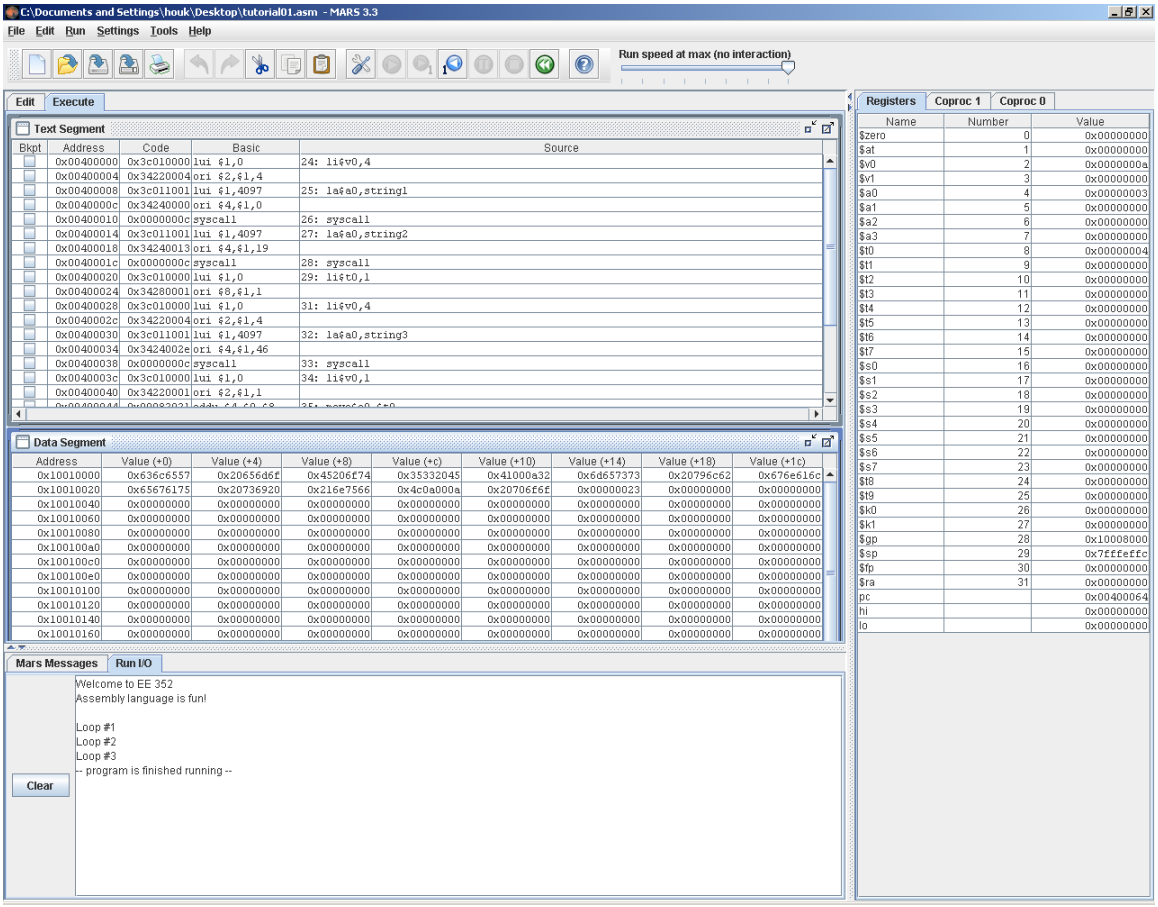


Figure 3 - MARS Simulator

3.2) The buttons at the top of the window can be used as shortcuts for the run menu. Use the “Reset” button to reset the program, and then try tracing through the program by clicking the step button.

3.3) You can adjust the speed the program runs by moving the slider to the right of the buttons. If you have an infinite loop in your program, it may be necessary to adjust (slow down) the speed of the simulator to prevent the MARS program from crashing.

Run the program. If a breakpoint has been set the program will stop at the next breakpoint.

Trace (Step) Into. Executes a single instruction. If the instruction is a procedure call (`jal`) the simulator will stop at the first instruction of the procedure.

Backstep. Undo the last step taken in the code.

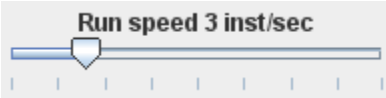
	Pause the currently running program. Press the run button to continue execution.
	Stop the currently running program. You will need to reset the simulator to execute the program again after stopping it.
	Reset. Resets the simulator, reinitializing the registers, program counter, and memory.
	Adjusts the speed that the simulator runs at.

Table 1 - Simulator Toolbar Commands

4. Using the Debugging Tools

4.1) When a program does not work as expected you will need to use the debugging tools provided with the simulator.

4.2) One of the primary tools used to debug a program is setting a breakpoint. You can break before execution of an instruction by clicking on the checkbox associated with each instruction on the far left of the execute window. Set a breakpoint at the instruction: `addi $t0,$t0,1`

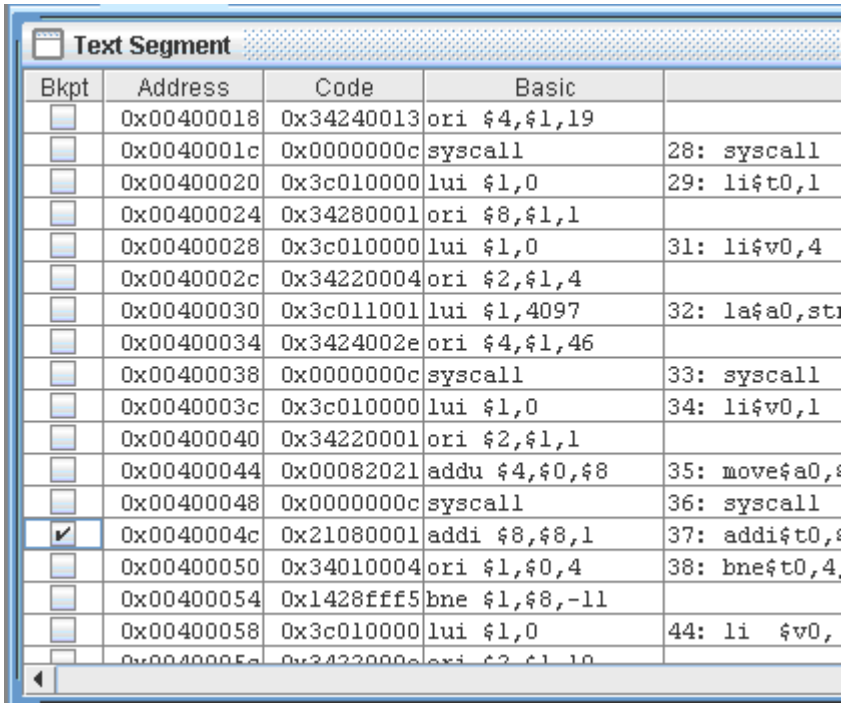


Figure 3 - MARS Breakpoints

4.3) Run the program until the breakpoint by clicking "Run". At this point in the program only the first loop iteration has been printed. (You will need to click back to the Run/IO window to see the output.)

4.4) Now use the "Trace Into" button to step through the loop that prints out the next line of text one character at a time. Step through the instructions until "Loop #2" is printed to the output window. Stop and find the value of the registers "t0" and

"pc" at that point? Has the line of code that the program counter points to executed yet?

4.5) The simulator also allows you to view the memory contents. The memory window appears in the middle of the screen and is titled "Data Segment". Remove the earlier breakpoint and add a breakpoint to line 33, "syscall". Click the run button so that the program executes up until the new breakpoint. We are now in the code right before "Loop #" is about to be printed for the third iteration. Notice that the $a0 register is now a pointer to the address where the "Loop #" text is stored. What is the memory location the register is pointing to?

4.6) Now look in the data segment area, and find the address $a0 points to. This is the memory section where the characters of the text "Loop #" is stored. Using an ASCII table find the address where the 'p' in "Loop" is located?

4.7) Exercise: Can you find where the word "Welcome" is stored in the memory?

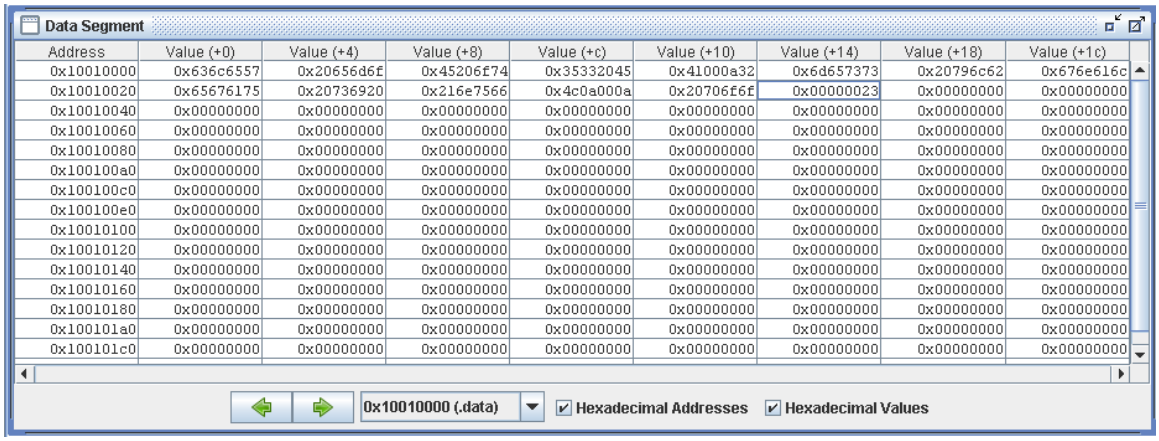


Figure 4 - Memory Window

American Standard Code for Information Interchange (ASCII)								
	$0b_6b_5b_4$							
$b_3b_2b_1b_0$	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7
0x0	NUL	DLE	SP	0	@	P	`	p
0x1	SOH	DC1	!	1	A	Q	a	q
0x2	STX	DC2	"	2	B	R	b	r
0x3	ETX	DC3	#	3	C	S	c	s
0x4	EOT	DC4	$	4	D	T	d	t
0x5	ENQ	NAK	%	5	E	U	e	u
0x6	ACK	SYN	&	6	F	V	f	v
0x7	BEL	ETB	'	7	G	W	g	w
0x8	BS	CAN	(	8	H	X	h	x
0x9	HT	EM	)	9	I	Y	i	y
0xA	LF	SUB	*	:	J	Z	j	z
0xB	VT	ESC	+	;	K	[	k	{
0xC	FF	FS	'	<	L	\	l	|
0xD	CR	GS	-	=	M	]	m	}
0xE	SO	RS	.	>	N	^	n	~
0xF	SI	US	/	?	O	-	o	DEL