

**CSC34300**

**Lecture 06: MIPS Assembly and  
MARS Simulator**

Instructor: Zheng Peng

Computer Science Department

City College of New York

# MARS

- **M**ips **A**ssembly and **R**untime **S**imulator
- A lightweight interactive development environment (IDE) for programming in MIPS assembly language
- Written in Java and requires Java Runtime Environment (JRE)
- Intended for educational user
- An open-source software
- <http://courses.missouristate.edu/KenVollmar/mars/index.htm>

# MARS Graphical User Interface

Demo\Fibonacci.asm - MARS 4.5

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Registers Coproc 1 Coproc 0

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x00001800
\$sp	29	0x00003ffe
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00000000
hi		0x00000000
lo		0x00000000

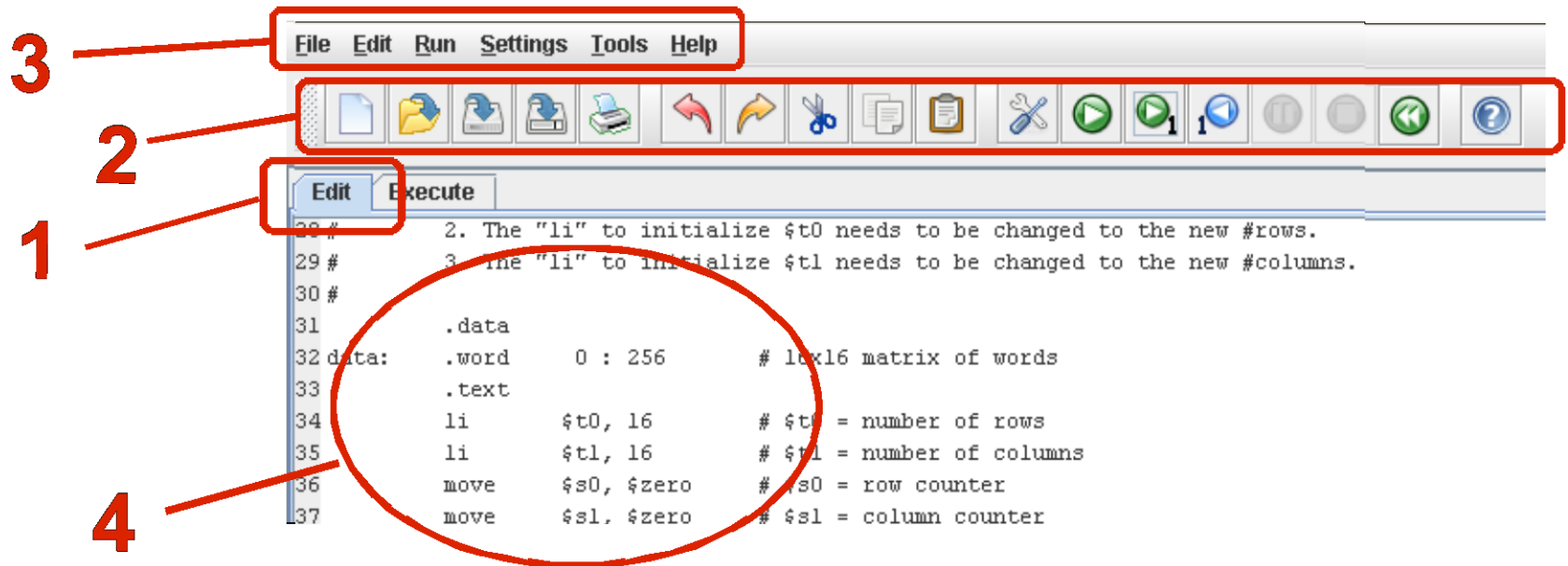
```
1 # Compute several Fibonacci numbers and put in array, then print
2 .data
3 fibs:.word 0 : 19      # "array" of words to contain fib values
4 size:.word 19          # size of "array" (agrees with array declaration)
5 prompt: .asciiz "How many Fibonacci numbers to generate? (2 <= x <= 19)"
6 .text
7     la $s0, fibs        # load address of array
8     la $s5, size         # load address of size variable
9     lw $s5, 0($s5)       # load array size
10
11 # Optional: user inputs the number of Fibonacci numbers to generate
12 #pr:  la $a0, prompt     # load address of prompt for syscall
13 #     li $v0, 4           # specify Print String service
14 #     syscall            # print the prompt string
15 #     li $v0, 0          # Replace this dummy with the correct numeric value????? # specify Read Integer service
16 #     syscall            # Read the number. After this instruction, the number read is in $v0.
17 #     bgt $v0, $s5, pr    # Check boundary on user input -- if invalid, restart
18 #     blt $v0, $zero, pr  # Check boundary on user input -- if invalid, restart
19 #     add $s5, $v0, $zero # transfer the number to the desired register
20
21     li $s2, 1           # 1 is the known value of first and second Fib. number
22     sw $s2, 0($s0)       # F[0] = 1
23     sw $s2, 4($s0)       # F[1] = F[0] = 1
24     addi $s1, $s5, -2    # Counter for loop, will execute (size-2) times
25
26 # Loop to compute each Fibonacci number using the previous two Fib. numbers.
27 loop: lw $s3, 0($s0)      # Get value from array F[n-2]
28      lw $s4, 4($s0)      # Get value from array F[n-1]
29      add $s2, $s3, $s4    # F[n] = F[n-1] + F[n-2]
30      sw $s2, 8($s0)      # Store newly computed F[n] in array
31      addi $s0, $s0, 4     # increment address to now-known Fib. number storage
32      addi $s1, $s1, -1    # decrement loop counter
33      bgtz $s1, loop      # repeat while not finished
34
35 # Fibonacci numbers are computed and stored in array. Print them.
```

Line: 1 Column: 1 ☒ Show Line Numbers

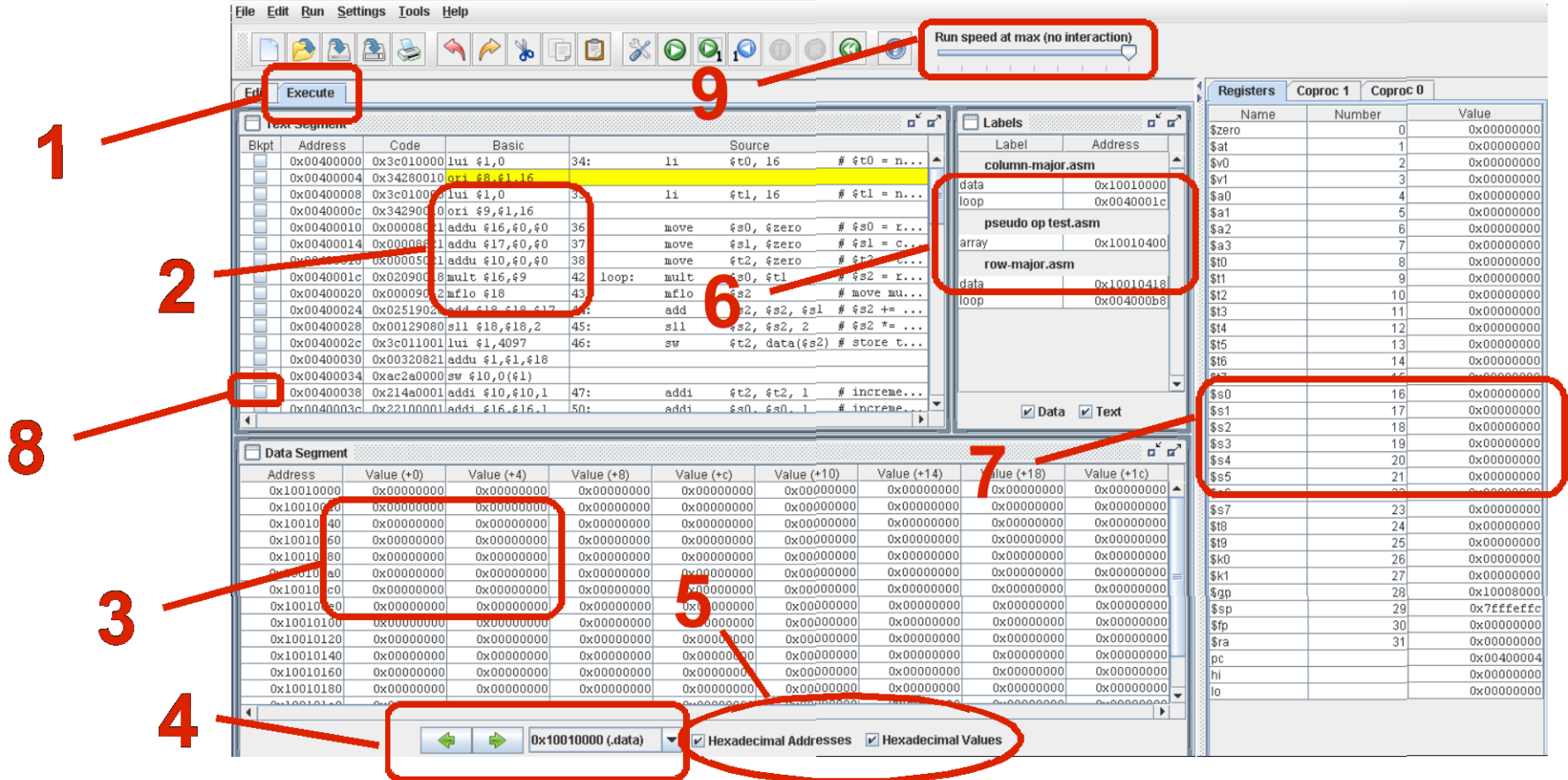
Mars Messages Run I/O

Clear

# MARS Main and Toolbar



1. Edit display is indicated by highlighted tab.
- 2, 3. Typical edit and execute operations are available through icons and menus, dimmed-out when unavailable or not applicable.
4. WYSIWYG editor for MIPS assembly language code.



1. Execute display is indicated by highlighted tab.
2. Assembly code is displayed with its address, machine code, assembly code, and the corresponding line from the source code file. (Source code and assembly code will differ when pseudoinstructions have been used.)
3. The values stored in Memory are directly editable (similar to a spreadsheet).
4. The window onto the Memory display is controlled in several ways: previous/next arrows and a menu of common locations (e.g., top of stack).
5. The numeric base used for the display of data values and addresses (memory and registers) is selectable between decimal and hexadecimal.
6. Addresses of labels and data declarations are available. Typically, these are used only when single-stepping to verify that an address is as expected.
7. The values stored in Registers are directly editable (similar to a spreadsheet).
8. Breakpoints are set by a checkbox for each assembly instruction. These checkboxes are always displayed and available.
9. Selectable speed of execution allows the user to "watch the action" instead of the assembly program finishing directly.

# Register List

- The right panel shows a list of registers
  - \$zero - always has zero value, READ ONLY
  - \$at - register is reserved for the assembler, DO NOT USE
  - \$t0-\$t9 - temporary registers, not preserved by subprograms
  - \$s0-\$s7 - general purpose registers, preserved by subprograms
  - \$k0-k1 - Reserved for kernel, DO NOT USE
  - \$sp - the stack pointer
  - \$fp - the frame pointer
  - \$ra - the return address
  - pc - program counter

# MIPS “Hello World!”

```
1  # MIPS "Hello World!" example
2
3      .data                # Data segment
4  msg: .asciiz "\nHello, World!\n"
5
6      .text                # Code segment
7      .globl main          # declare main to be global
8  main:
9      li $v0, 4             # select a system call to print a string
10     la $a0, msg           # load address of string to be printed into $a0
11     syscall              # make the system call to perform operation
12
13     li $v0, 10            # terminate program
14     syscall
```

# MIPS Directives\*

- Assembler **directives** are instructions that direct the assembler to do something.
- Directives do many things: some tell the assembler to set aside space for variables, others tell the assembler to include additional source files, and others establish the start address for your program.

.align n	Align the next datum on a $2^n$ byte boundary.
.ascii str	Store the string in memory, but do not null-terminate it.
.asciiz str	Store the string in memory and null-terminate it.
.data <addr>	The following data items should be stored in the data segment. If the optional argument <i>addr</i> is present, the items are stored beginning at address <i>addr</i> .
.globl sym	Declare that symbol sym is global and can be referenced from other files.
.space n	Allocate <i>n</i> bytes of space in the current segment
.text <addr>	The next items are put in the user text segment. If the optional argument <i>addr</i> is present, the items are stored beginning at address <i>addr</i> .

\*Reference: [http://students.cs.tamu.edu/tanzir/csce350/reference/assembler\\_dir.html](http://students.cs.tamu.edu/tanzir/csce350/reference/assembler_dir.html)



# SYSCALL functions in MARS (1)

- A number of system services, mainly for input and output, are available for use by your MIPS program.
- How to use SYSCALL system services
  - Step 1. Load the service number in register \$v0.
  - Step 2. Load argument values, if any, in \$a0, \$a1, \$a2, etc.
  - Step 3. Issue the SYSCALL instruction.
  - Step 4. Retrieve return values, if any, from result registers as specified.

All information are from:

<http://courses.missouristate.edu/kenvollmar/mars/help/syscallhelp.html>

# SYSCALL functions in MARS (2)

Service	\$v0	Arguments	Result
print integer	1	\$a0 = integer to print	
print string	4	\$a0 = address of null-terminated string to print	
read integer	5		\$v0 contains integer read
read string	8	\$a0 = address of input buffer \$a1 = maximum number of characters to read	
terminate execution	10		

Complete list available at:

<http://courses.missouristate.edu/kenvollmar/mars/help/syscallhelp.html>

# SYSCALL functions in MARS (3)

- Example:
  - Display the value stored in \$t0 on the console

```
2  li $v0, 1           # service 1 is print integer
3  add $a0, $t0, $zero # load desired value into argument
4                          # register $a0, using pseudo-op
5  syscall              # make the system call
```

- Read a string from the console

```
1      .data
2  str: .byte 20
3
4      .text
5      .globl main
6  main:
7      la $a0, str      # $a0, buffer address
8      li $a1, 20       # $a1, size of the buffer
9      li $v0, 8        # read a string
10     syscall
11
12     la $a0, str      # $a0, buffer address
13     li $v0, 4        # print a string
14     syscall
15
16     li $v0, 10       # terminate program
17     syscall
```

# Another MIPS Programming Example

- Generating Fibonacci Numbers:
  - Fibonacci number: every number after the first two is the sum of the two preceding ones: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...
  - $F_n = F_{n-1} + F_{n-2}$
  - In a HLL the such a function might resemble the following

```
void PrintTwelveFibonacci()
{
    int Fibonacci[9];
    Fibonacci[0] = 1;
    Fibonacci[1] = 1;

    for (int k = 2; k <= 9; k++)
        Fibonacci[k] = Fibonacci[k - 1] + Fibonacci[k - 2];

    for (int k = 0; k <= 9; k++)
        print(Fibonacci[k]);
}
```

# Another MIPS Programming Example

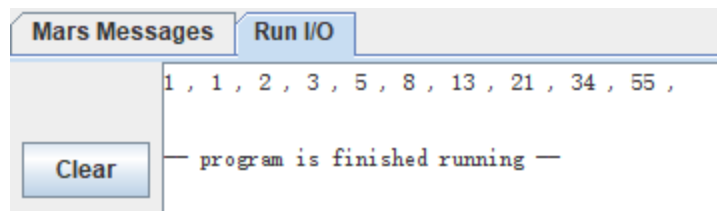
```
1      .data                # data segment
2  nl:   .ascii "\n"        # ASCII for a new line
3      .align 2             # aligned at word boundary
4  comma: .ascii ", "       # note the space
5      .align 2
6
7      .text                # Code segment
8      .globl main          # declare main to be global
9  main: # $t2: F(n-2), $t1: F(n-1), $t0: F(n)
10     li    $t2, 0          # $t2=0; initial value of F(n-2)
11                        # in this case, F(0)
12     li    $t1, 1          # $t1=1; initial value of F(n-1)
13                        # in this case, F(1)
14     # -----
15     move   $a0, $t1        # $a0= $t1, which is F(1)
16     li     $v0, 1          # system call, type 1, i.e. integer
17     syscall                # make system call to print the value of F(1)
18     # -----
19     la     $a0, comma      # $a0 = address of comma
20     li     $v0, 4          # system call, type 4, i.e. string
21     syscall                # make system call to print string
22     # -----
23     li     $t3, 9          # $t3 is the counter to be decremented
```

# Another MIPS Programming Example

```
24  loop:
25      add    $t0,$t1,$t2 # F(n) = F(n-1) + F(n-2)
26      # -----
27      move    $a0,$t0    # $a0= $t0, which is F(n)
28      li      $v0,1      # system call, type 1, i.e. integer
29      syscall                      # print the value of F(n)
30      # -----
31      la      $a0,comma    # $a0 = address of comma
32      li      $v0,4        # system call, type 4, i.e. string
33      syscall                      # make system call to print string
34      # -----
35      addi    $t4,$t4,1    # $t4 = $t4 + 1; increment n
36      move    $t2,$t1      # $t2 = $t1; previous F(n-1) becomes F(n-2)
37      move    $t1,$t0      # $t1 = $t0; previous F(n) becomes F(n-1)
38      addi    $t3,$t3,-1   # $t3 = $t3 - 1; decrement the counter
39      bne     $0,$t3,loop  # continue if $t3 is not 0
40  Exit:  # -----
41      la      $a0,nl      # $a0= address of "nl"
42      li      $v0,4        # system call, type 4, print an string
43      syscall                      # print a newline
44      # -----
45      li      $v0,10       # System call, type 10, standard exit
46      syscall                      # ...and call the OS
```

# Another MIPS Programming Example

- To run the program, press F3 and then F5 in MARS
- Then the output, if any, will show up at the bottom window of the MARS simulator



# Assembling the code

- Once this code is loaded into MARS we can assemble it ... click Run > Assemble or press F3
  - Our window will change to display the Text and Data segments

The screenshot shows the MARS interface with the 'Execute' tab selected. The 'Text Segment' window displays the assembly code for a Fibonacci sequence calculation. The 'Data Segment' window shows the memory layout for the program, with addresses ranging from 0x10010000 to 0x10010100. The 'Data Segment' window is currently displaying the 'Value (+0)' column for each address.

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x3c011001	lui \$1,0x00001001	6: la \$t0, fibs # load address of array
<input type="checkbox"/>	0x00400004	0x34280000	ori \$8,\$1,0x00000000	
<input type="checkbox"/>	0x00400008	0x3c011001	lui \$1,0x00001001	7: la \$t5, size # load address of size variable
<input type="checkbox"/>	0x0040000c	0x342d0030	ori \$13,\$1,0x00000030	
<input type="checkbox"/>	0x00400010	0x8dad0000	lw \$13,0x00000000(\$13)	8: lw \$t5, 0(\$t5) # load array size
<input type="checkbox"/>	0x00400014	0x240a0001	addiu \$10,\$0,0x0000...	9: li \$t2, 1 # 1 is first and second Fib. number
<input type="checkbox"/>	0x00400018	0x46241000	add.d \$f0,\$f2,\$f4	10: add.d \$f0, \$f2, \$f4
<input type="checkbox"/>	0x0040001c	0xad0a0000	sw \$10,0x00000000(\$8)	11: sw \$t2, 0(\$t0) # F[0] = 1
<input type="checkbox"/>	0x00400020	0xad0a0004	sw \$10,0x00000004(\$8)	12: sw \$t2, 4(\$t0) # F[1] = F[0] = 1
<input type="checkbox"/>	0x00400024	0x21a9fffe	addi \$9,\$13,0xffffffff	13: addi \$t1, \$t5, -2 # Counter for loop, will execute (size-...
<input type="checkbox"/>	0x00400028	0x8d0b0000	lw \$11,0x00000000(\$8)	14: loop: lw \$t3, 0(\$t0) # Get value from array F[n]

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x0000000c	0x68540020	0x69462065	0x616e6f62
0x10010040	0x20696363	0x626d756e	0x20737265	0x3a657261	0x0000000a	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Navigation buttons: ⬅️ ➡️

Address: 0x10010000 (.data) ▼

Options: ☒ Hexadecimal Addresses ☒ Hexadecimal Values ☐ ASCII



# Execution – Single Step

- We can click Run > Step or press F7 to single step through the code
- As you step through the code you will notice the line you are about to execute is highlighted in yellow, and the value of the program counter matches the address value of that line

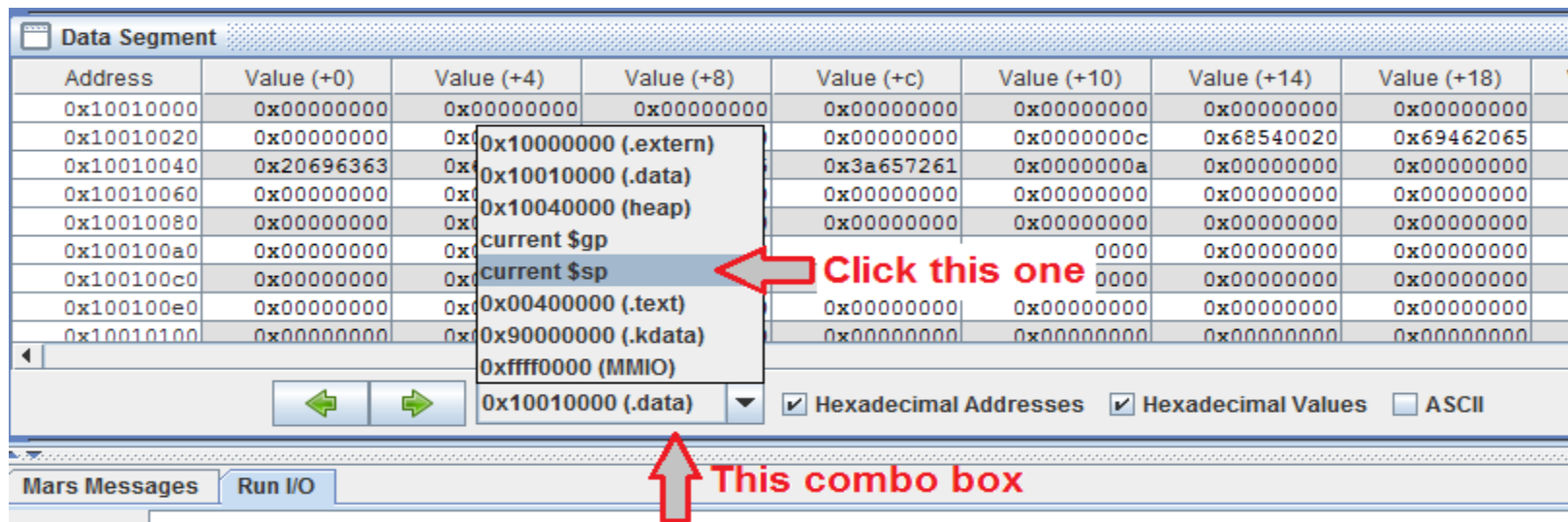
<input type="checkbox"/>	0x00400008	0x3c011001	lui \$1,0x00001001	7:	la \$t5, size
<input type="checkbox"/>	0x0040000c	0x342d0030	ori \$13,\$1,0x00000030		
<input type="checkbox"/>	0x00400010	0x8dad0000	lw \$13,0x00000000(\$13)	8:	lw \$t5, 0(\$t5)
<input type="checkbox"/>	0x00400014	0x240a0001	addiu \$10,\$0,0x0000...	9:	li \$t2, 1
<input type="checkbox"/>	0x00400018	0x46241000	add.d \$f0,\$f2,\$f4	10:	add.d \$f0, \$f2, \$f4
<input type="checkbox"/>	0x0040001c				
	\$fp	30	0x00000000	(\$t0)	
	\$ra	31	0x00000000		
	pc		0x00400014		

- You may have also noticed in the registers window registers are being highlighted in green, this is to indicate the contents of that register have changed

\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x0000000c
\$t6	14	0x00000000
\$t7	15	0x00000000

# Execution – Viewing the Stack

- To see the data currently on the stack, click the combo box in the Data Segment window, and change the value to **current \$sp**



- This fibonacci.asm does not use the stack at all, so the view will revert back to data as you step through the code. So we need to rewrite the code so that it incorporates the stack – or you can download it from the course website