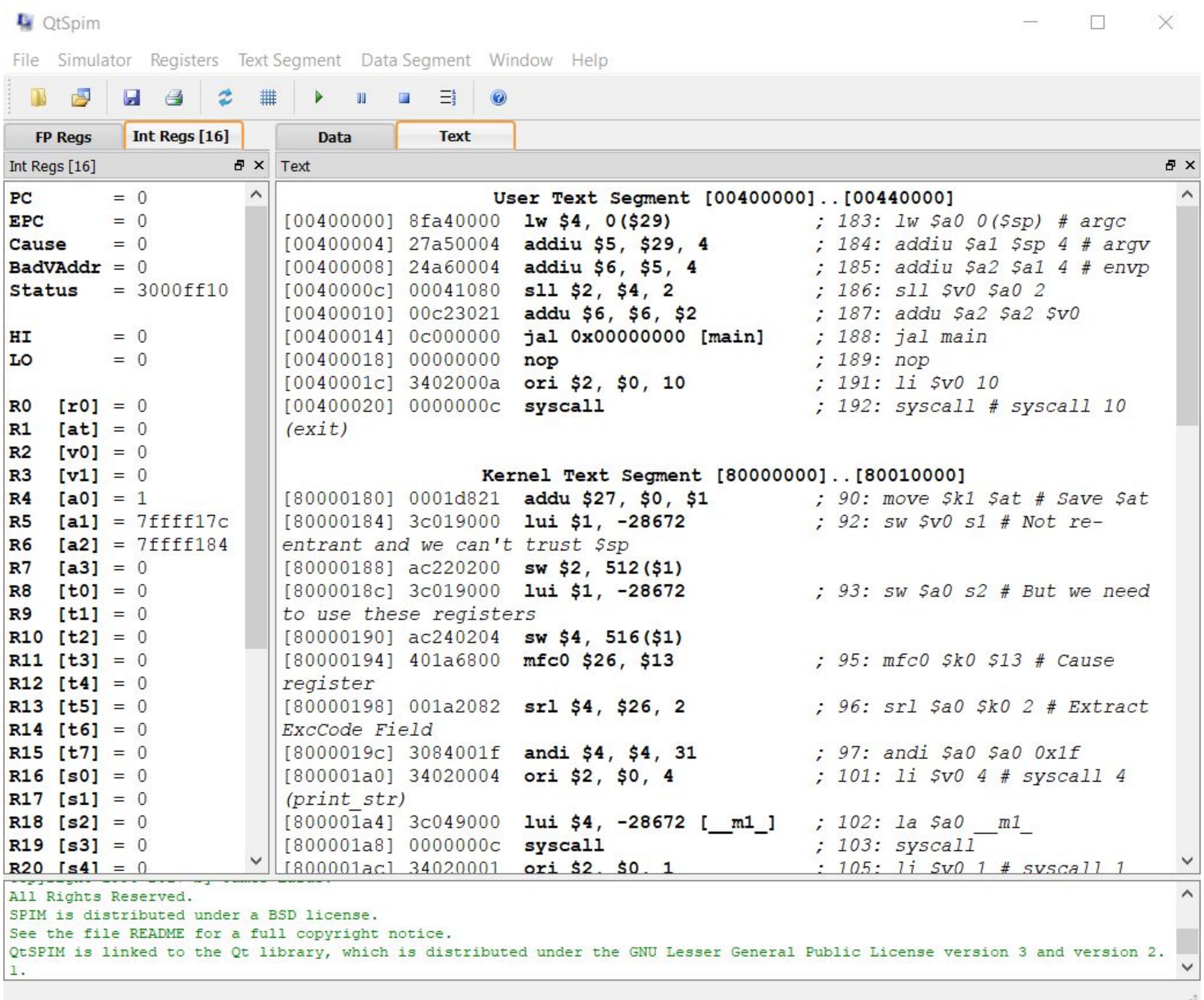
Week 9 Recitation
MIPS Environment Setup + Program Overview
(<u>NEXT WEEK</u>: *Project Breakdown and File Reading*)

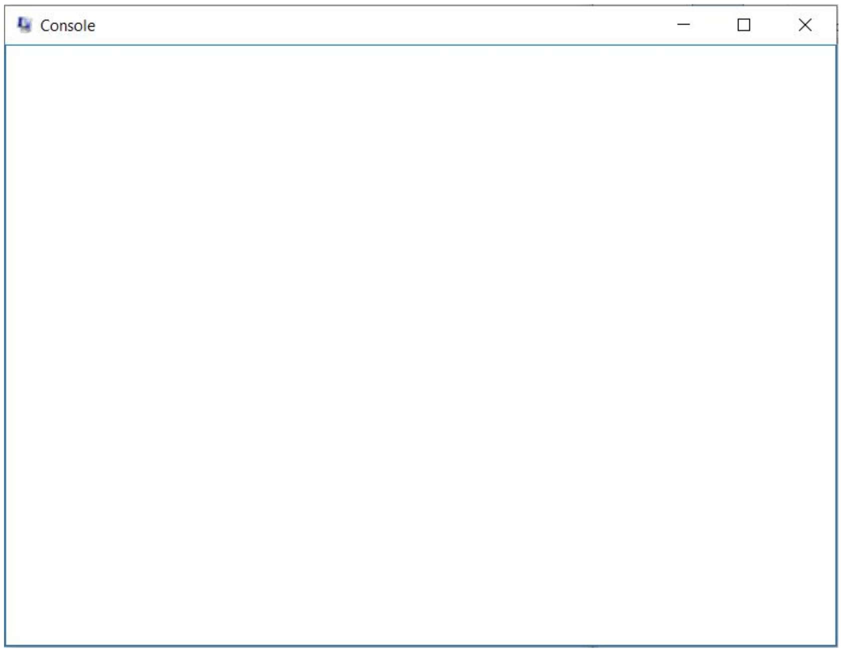
<u>Install MIPS Simulator QtSpim</u>
(1) Goto: http://spimsimulator.sourceforge.net/
Press “Download Spim”
Download Latest Version listed for your computer.
i. If you have issues downloading the latest version for windows, try to install the “QtSpim_9.1.18_Windows.exe.”

(2) Upon launch, you should see this interface:

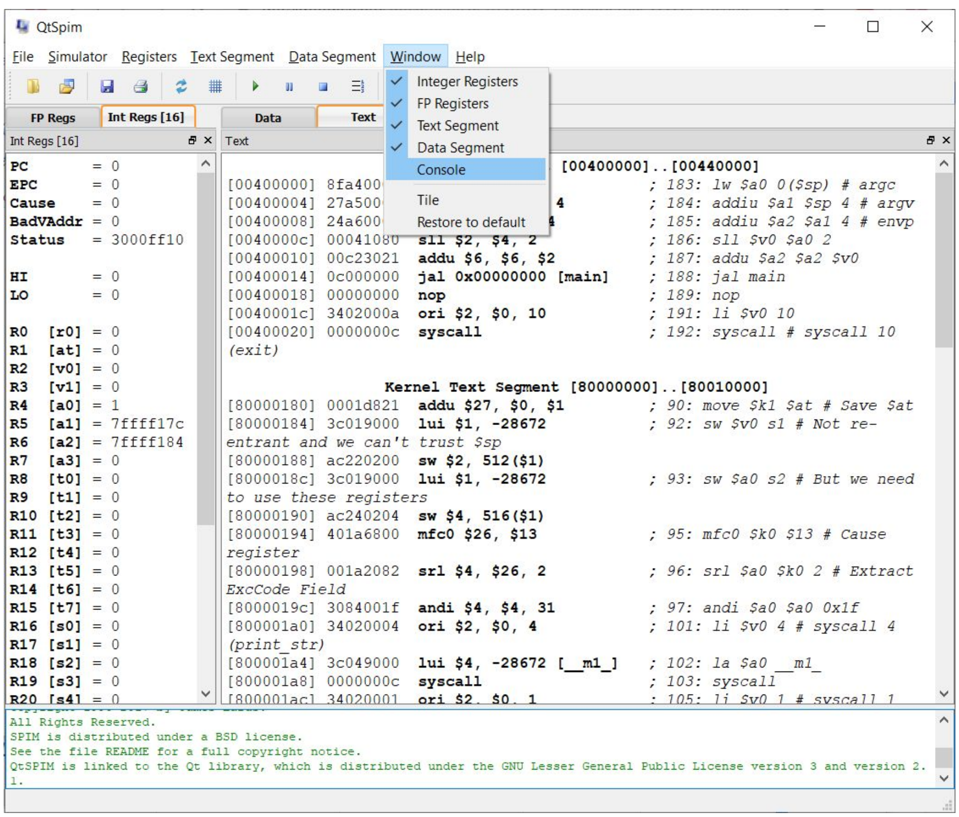


Using QtSpim

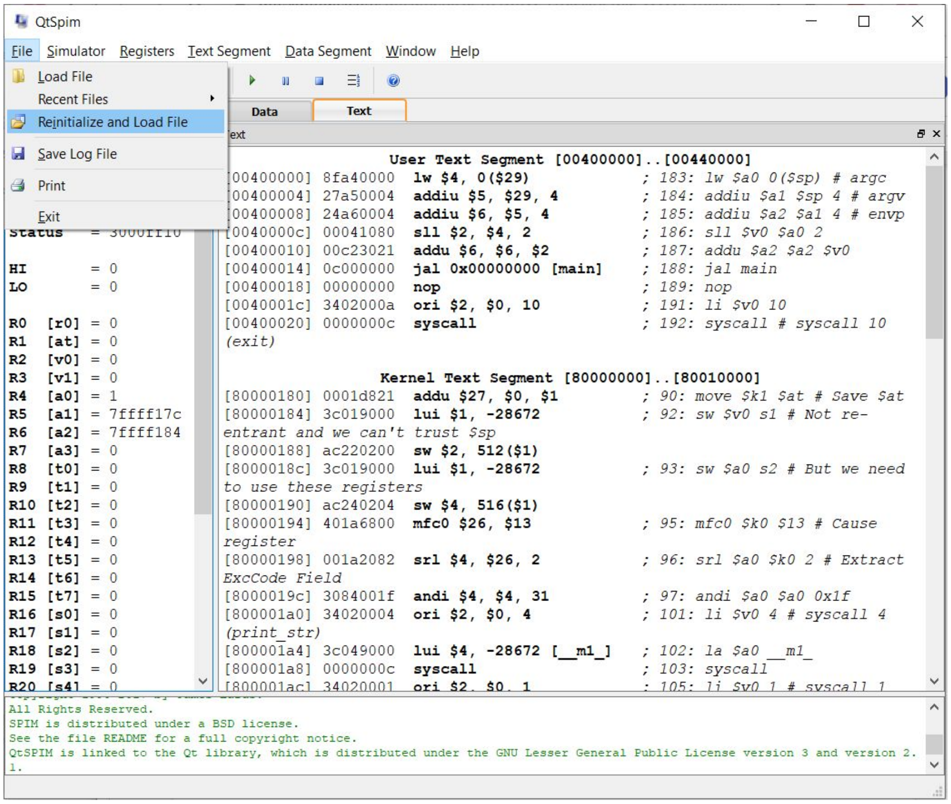
(1) Outputs using syscall and sys flags will be displayed in your “Console” window.



(2) If you do not see your “Console” you can open the window back up like so:



(3) In order to run files, you must be sure to “reinitialize and load file” from your directory. Files must be in “.asm” format.



(4) Then you hit the play button to run your code.

Checking if setup is correct

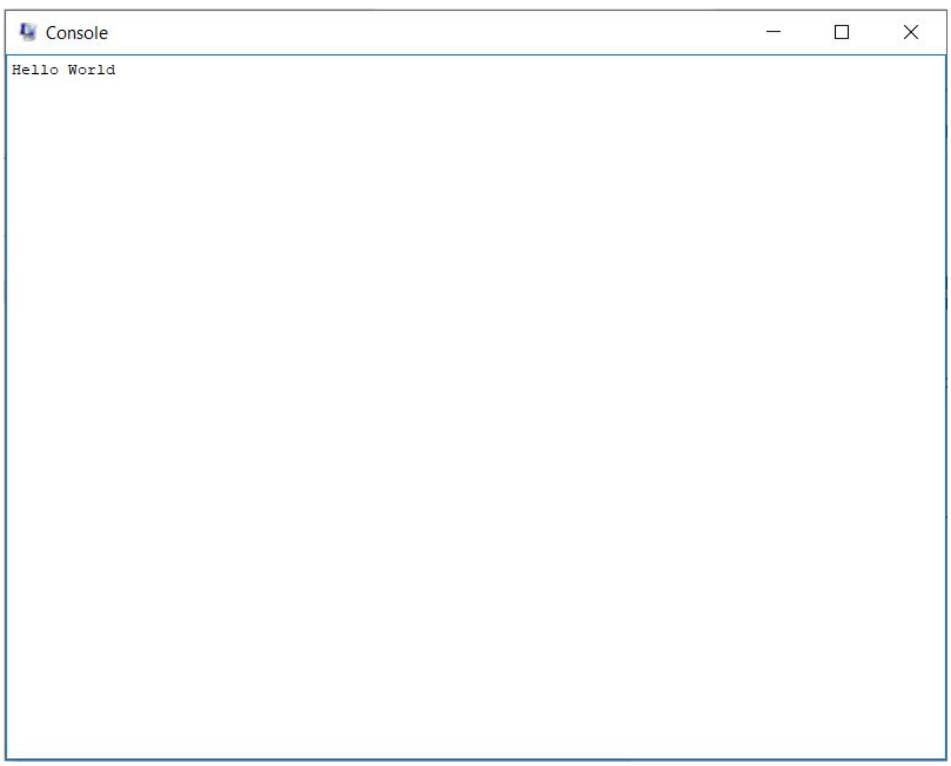
(1) Make the following code into an .asm file and run it:

```
.data
        hello: .asciiz "Hello World\n"
.text
j main

main:
        li $v0, 4
        la $a0, hello
        syscall

        li $v0,10
        syscall
```

(2) Following the steps to run a program in QtSpim, you should see this in your Console



SYSCALL FLAGS

Service	System call code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		
print_char	11	$a0 = char	
read_char	12		char (in $v0)
open	13	$a0 = filename (string), $a1 = flags, $a2 = mode	file descriptor (in $a0)
read	14	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars read (in $a0)
write	15	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars written (in $a0)
close	16	$a0 = file descriptor	
exit2	17	$a0 = result	

REGISTERS

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	No

Directives

They are used to separate your code (in the case of .data and .text) and allocate memory for variables and strings.

Directives to know:

Directive	Format	Usage
.word	tag : .word w1, … , wn	Stores values in 32-bit or 4-byte quantities
.space	tag : .space num_of_bytes	Allocates set space of memory.
.float	tag : .float f1, … , fn	Stores floating point numbers in memories
.align	.align num	Aligns data to 2^{num} boundary (example later)
.asciiz	tag: .asciiz str	Stores a string str.

Tags in Directives

la is used to access the tags in our directives. This is because these directive tags serve to hold the memory address of these memory allocations.

So if we do something like add one to the address, we should expect to print "ello World" instead:

```
.data
        hello: .asciiz "Hello World\n"

.text
j main

main:
        li $v0, 4
        la $a0, hello
        addi $a0, $a0, 1
        syscall

        li $v0,10
        syscall
```

Since these tags are actually holding memory addresses, we can make use of la, and treat these memory allocations like they are arrays in memory.

Word Arrays and Alignment
When you store words into memory, because they take up a 4 byte value, we must align the memory we allocate to a position divisible by 4 (e.g. 0x0004, instead of 0x0003). To specify this alignment, we must use .align 2.

```
.data
        .align 2
        array_x: .space 2048

.text
        la $t0, array_x
        lw $t2, 4($t0)        # array_x[1] = $t2
```

Looping in Mips
Loops in MIPS follow a general structure of:

[Tag for Loop]:
        [Check for exit condition
        Typically using bgt, blt, bgte, blte, beq]
 	
        [Code to inductively solve problem]
 	
        [Update necessary variables]
 	
        j [Tag for Loop]
 	
[Tag for Outside of Loop]
	Loop:
	        blt $t0, $zero, Outside_Loop
	        beq $t0, $zero, Outside_Loop
	
	        add $t1, $t1, $t0
	
	        addi $t0, $t0, -1
	
	        j Loop
	
	Outside_Loop:

DEMONSTRATIONS
Write a MIPS program that will take some positive integer input. Then count-down to zero from that input.

E.g. Input is 10
Loop down and print all values from 10 -> 0

Approach:

1. Take in an input from the user.
2. Do some minimal input-validation for the input.
   I.e. input must be greater than or equal to zero.
3. Loop from that value and print every loop.

Write a MIPS program that can print out the first n multiples of 5.

Approach:

The same exact thing as before, except you multiply by 5 before you print.

TRY AT HOME

Write a MIPS program for a 1x5 matrix addition calculator.

Input: Accept 10 random integer inputs from the user.
Output: The matrix sum of the 10 integer inputs.

E.g.

1	**2**	**5**	**6**	**1**

+

6	**2**	**3**	**122**	**34**

|

V

7	**4**	**8**	**128**	**35**

Approach**:**

Set up how we want to store the input and output.
Set up taking in the input.

Loop through and calculate each cell of the output.

At the heart of it: it's pretty much

for (i = 0; i < 5; i ++) { C[i] = A[i] + B[i]; }