CS 3330 Computer Architecture, Spring 2020
Lab 1: Introduction to MIPS Simulator: "QtSPIM"

Instructor: Prof. Samira Khan
TAs: Amel Fatima, Sihang Liu, Korakit Seemakhupt, Yasas Senerivatne,Yizhou Wei,
Min Jae Lee, Nikita Semichev, Yuying Zhang.

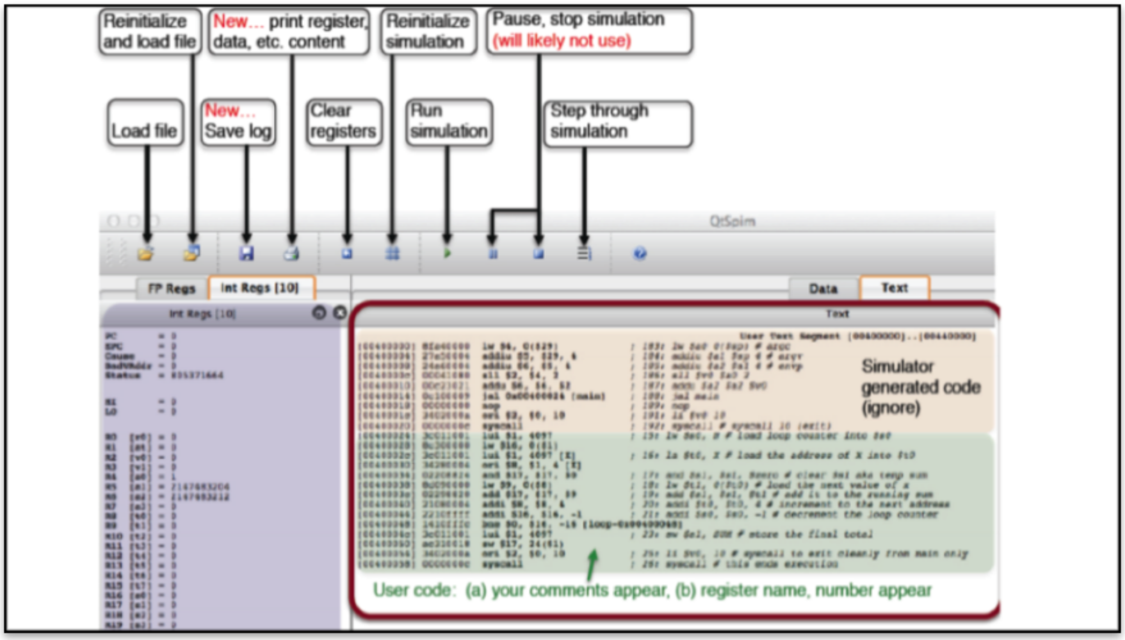
Assigned: Wed., 1/29, 2020
Due: **Wed., 2/5, 2020**

Installation of QtSPIM [10 points]

SPIM, which is just MIPS spelled backwards, is a software that simulates the working of a MIPS processor. It is capable of running MIPS32 assembly language programs and it allows users to inspect the various internal states of a MIPS processor.

The most up-to-date version of the SPIM simulator, called "QtSPIM", is maintained by James Larus, a former Professor at the University of Wisconsin (now at EPFL). It is free and maintained on SourceForge. Please follow the steps below to complete the installation of this MIPS simulator:

1. Navigate to the SourceForge website "`https://sourceforge.net/projects/spimsimulator/files/`"
2. You will see the choice of a number of SPIM downloads. Select the relevant file according to your Operating System and wait for it to complete the download (if v9.1.21 does not work on your system, try an older version).
3. The download software will ask if you want to open or save. Click "save." Most browsers have a space for "download files," so save it there.
4. Once the program is saved in the download folder, double click on that program. Note that it is an .exe file, so it will start to install as soon as you click on it.
5. The program will ask to create a folder named "C:/ProgramFiles/QtSpim"to store SPIM in that.
6. Once QtSpim is created, you may open it to see if you get the register and text windows. Note that you have to click on a tab at the top to get the "Data" window. If you can see these windows, then QtSPIM should be correctly installed.
7. If you want to run the same program multiple times, then you should clear the registers each time before running the program again. There is an icon named "Clear Registers" shown in the figure below. Remember to use that before running the same program again.
8. If you want to run a different assembly program, then you should first clear the registers and reinitialize your simulator before loading a different program. The icon to reinitialize your simulator is shown in the figure below.
9. If your "simulator's console" shuts down while running a program, then you can always reopen it by pressing the "windows" tab and clicking on the console option.



The graphical interface of your SPIM simulator should look like the screenshot shown above. We will play around with it to get a better understanding of all available cool features that this simulator provides us. Please take a look at the labeled icons and see if you can find them in your installed version of the SPIM simulator

Task1 [20 points]

The "Data tab" shows the content of the Data memory space. This content includes the variables and array data you create, along with the stack content. This task will test your understanding of the "Data Segment" in the SPIM simulator. Create a ".asm" file with the MIPS code given below and then click on the "load file" icon to load the created assembly file in your Simulator.

```
# # # # # # # # # # # # # #
#  CS3330: Lab1.
#
#  Task1: Demonstration
#  of ASCII data stored
#  in Memory (Data Segment)
#
#  Filename: Lab1.asm
#
       #The Data Segment
       .data
a1:             .ascii              "We Welcome"
a2:             .ascii              "You To"
a3:             .ascii              "CS3330"

       #The Text Segment
       .text

main:    #Normal termination of Program.
         li  $v0, 10
         syscall
```

Take a close look at the content of the memory address "0x10010000" in the data segment. Can you figure out where and how are the strings stored in the memory? (Hint: Write out the ASCII values in hexadecimal form of the characters to find their mapping in the memory). Keep in mind that "space" also has a hexadecimal value in the ASCII chart. Please write down the characters that the following memory locations hold in the data segment (Each memory location holds one byte, so memory locations from "10010000-10010003" will hold 4 bytes of data. Remember that each character occupies one byte of space in memory).

1. "10010000-10010003":_________________
2. "10010008-1001000b":_________________
3. "10010010-10010013":_________________
4. "1001000c-1001000f":_________________

Task2 [30 points]

Please follow the link "`http://www.cs.virginia.edu/~smk9u/CS3330S20/task2.s`" to download the MIPS file named "task2.s". Download that code and run it in your simulator. You should run the code line by line so that you can see the changes in the values of the registers. The "single-step execution" mode will help you solve this task and identify the changes in the values of the registers. (Remember to clear your registers and reinitialize your simulator before loading a new assembly program in your simulator)

i. The value of the register ``t2'' changes eighteen times during the execution. List all the values that the register `t2` acquires during the execution in a chronological order. Use the single-step execution mode to answer this question.

ii. Set a "breakpoint" at the instruction ``addi $29, $29, 4" and record the value of the Program Counter (PC), register ``t0'', ``t4'', and ``t5'' at that point. Please do not exceed your execution beyond the breaking point.

iii. In what memory location is the instruction ``lw $12, -4($10)" stored?

Task3 [40 points]

This task involves your interaction with the MIPS code. You will use the "SPIM console" to enter two binary numbers and check the corresponding output. Please download the file "task3.asm" by following the link "`http://www.cs.virginia.edu/~smk9u/CS3330S20/task3.asm`" and run it in your simulator. You will enter two similar length binary numbers through the console and it will display the output for you.

i. Enter two binary inputs (same length) of your choice in the console and observe the output. List a combination of inputs that you use and the outputs that you observe in the table below.

Input1	Input2	Output

ii. What is the MIPS program doing with the two binary numbers? How is the output calculated?

iii. How much user memory is reserved and is made unavailable for use? You can run both the tasks to check which portion of the memory is never allocated to the program and is reserved. You only need to check the "User data segment" in the data segment tab to find out the answer.(Remember to give your answer in Kilo bytes).

iv. Refer to the data segment and calculate the total user memory that is available for use. (First calculate the total user memory and then subtract the reserved memory to get the answer.)

Bonus Question [20 points]

1. Run your assembly code for "Fibonacci Numbers" from HW1 and record the output from the console. Take a screenshot and paste it in your lab1's solution to score the bonus points.

Handin

You should electronically hand in your assignment (in pdf format) to Collab.

For your Reference:

Please find the attached ASCII table on the next page for your reference. All you need is the "Hex" value of the corresponding ASCII symbol from the table.

Dec	Hex	Oct	Binary	Char	Dec	Hex	Oct	Binary	Char	Dec	Hex	Oct	Binary	Char	Dec	Hex	Oct	Binary	Char
0	00	000	0000000	NUL (null character)	32	20	040	0100000	space	64	40	100	1000000	@	96	60	140	1100000	`
1	01	001	0000001	SOH (start of header)	33	21	041	0100001	!	65	41	101	1000001	A	97	61	141	1100001	a
2	02	002	0000010	STX (start of text)	34	22	042	0100010	"	66	42	102	1000010	B	98	62	142	1100010	b
3	03	003	0000011	ETX (end of text)	35	23	043	0100011	#	67	43	103	1000011	C	99	63	143	1100011	c
4	04	004	0000100	EOT (end of transmission)	36	24	044	0100100	$	68	44	104	1000100	D	100	64	144	1100100	d
5	05	005	0000101	ENQ (enquiry)	37	25	045	0100101	%	69	45	105	1000101	E	101	65	145	1100101	e
6	06	006	0000110	ACK (acknowledge)	38	26	046	0100110	&	70	46	106	1000110	F	102	66	146	1100110	f
7	07	007	0000111	BEL (bell (ring))	39	27	047	0100111	'	71	47	107	1000111	G	103	67	147	1100111	g
8	08	010	0001000	BS (backspace)	40	28	050	0101000	(	72	48	110	1001000	H	104	68	150	1101000	h
9	09	011	0001001	HT (horizontal tab)	41	29	051	0101001	)	73	49	111	1001001	I	105	69	151	1101001	i
10	0A	012	0001010	LF (line feed)	42	2A	052	0101010	*	74	4A	112	1001010	J	106	6A	152	1101010	j
11	0B	013	0001011	VT (vertical tab)	43	2B	053	0101011	+	75	4B	113	1001011	K	107	6B	153	1101011	k
12	0C	014	0001100	FF (form feed)	44	2C	054	0101100	,	76	4C	114	1001100	L	108	6C	154	1101100	l
13	0D	015	0001101	CR (carriage return)	45	2D	055	0101101	-	77	4D	115	1001101	M	109	6D	155	1101101	m
14	0E	016	0001110	SO (shift out)	46	2E	056	0101110	.	78	4E	116	1001110	N	110	6E	156	1101110	n
15	0F	017	0001111	SI (shift in)	47	2F	057	0101111	/	79	4F	117	1001111	O	111	6F	157	1101111	o
16	10	020	0010000	DLE (data link escape)	48	30	060	0110000	0	80	50	120	1010000	P	112	70	160	1110000	p
17	11	021	0010001	DC1 (device control 1)	49	31	061	0110001	1	81	51	121	1010001	Q	113	71	161	1110001	q
18	12	022	0010010	DC2 (device control 2)	50	32	062	0110010	2	82	52	122	1010010	R	114	72	162	1110010	r
19	13	023	0010011	DC3 (device control 3)	51	33	063	0110011	3	83	53	123	1010011	S	115	73	163	1110011	s
20	14	024	0010100	DC4 (device control 4)	52	34	064	0110100	4	84	54	124	1010100	T	116	74	164	1110100	t
21	15	025	0010101	NAK (negative acknowledge)	53	35	065	0110101	5	85	55	125	1010101	U	117	75	165	1110101	u
22	16	026	0010110	SYN (synchronize)	54	36	066	0110110	6	86	56	126	1010110	V	118	76	166	1110110	v
23	17	027	0010111	ETB (end transmission block)	55	37	067	0110111	7	87	57	127	1010111	W	119	77	167	1110111	w
24	18	030	0011000	CAN (cancel)	56	38	070	0111000	8	88	58	130	1011000	X	120	78	170	1111000	x
25	19	031	0011001	EM (end of medium)	57	39	071	0111001	9	89	59	131	1011001	Y	121	79	171	1111001	y
26	1A	032	0011010	SUB (substitute)	58	3A	072	0111010	:	90	5A	132	1011010	Z	122	7A	172	1111010	z
27	1B	033	0011011	ESC (escape)	59	3B	073	0111011	;	91	5B	133	1011011	[	123	7B	173	1111011	{
28	1C	034	0011100	FS (file separator)	60	3C	074	0111100	<	92	5C	134	1011100	\	124	7C	174	1111100	|
29	1D	035	0011101	GS (group separator)	61	3D	075	0111101	=	93	5D	135	1011101	]	125	7D	175	1111101	}
30	1E	036	0011110	RS (record separator)	62	3E	076	0111110	>	94	5E	136	1011110	^	126	7E	176	1111110	~
31	1F	037	0011111	US (unit separator)	63	3F	077	0111111	?	95	5F	137	1011111	_	127	7F	177	1111111	DEL