Name:

Lab for ITSC 3181, Introduction to Computer Architecture, Spring 2023

Lab #04 and #05: Converting C programs to RISC-V assembly and simulate their execution using RARS simulator. Due on 02/10 and count for 3% of the final grade percentage.

			Lab	#04 and #	05		
Steps	2	3	4	5	6	7 (bonus)	Total
Total Points	10	20	20	20	30	20	120
Your points							

We will use RISC-V Assembler and Runtime Simulator (RARS) for this lab, which is available from <u>https://github.com/TheThirdOne/rars.</u> A video for introducing how to use RARS simulator is available from <u>https://passlab.github.io/ITSC3181/resources/UsingRARS_ITSC3181.mp4</u> or from Canvas Medial Gallery.

- Download the latest RARS jar file from <u>https://github.com/TheThirdOne/rars/releases/download/v1.5/rars1_5.jar</u> on your computer. You should then be able to launch the jar file by either double clicking it or from launcher such as using open command in Mac OS X terminal.
- 2. Create and execute the hello world program following instructions from <u>https://github.com/TheThirdOne/rars/wiki/Creating-Hello-World</u>. Spend some time to read and understand each line of that page. Then in RARS, create the Hello World program, and then assembly and run the program by clicking the sub-menu items of the Run menu. Play with the example, menu items and the RARS interface to get familiar with the RARS tool. Check the address, binary code, instructions and source of the assembled code, and also check the register values and memory values (data segment part) of the program execution. After you run the program multiple times, you should run it step-by-step, i.e. instruction by instruction, and observe the change of values in registers and other locations. There are two sections, named .text section and .data section, in the program. The .text section and "str" label in the .data section. The two labels are symbols representing the addresses of the memory locations where corresponding data or code are stored in memory.

	A A A	5 🕈 🐌	3 🖸 🥙 🚿			Run speed at max							
					Edit Execute					Registers	Floating Point	Contro	and Statu
			Tex	kt Segment				eee Lab	ls	Name	Number	Value	
-								Label A	ddress 🔺	zero		0	0x0000x0 0x0000x0
Program	Arguments:							hello.asm		Sp		2	0x7fffe
Bkpt A	ddress Code	Basic	Source					main	0×08480808	gp		3	0×10008
		100513 addi x10,x0,				e and we want to lo		str	0×10010000	tp		4	0×000006
		c10597 auipc x11,0x		a al, str # la is s	imilar to li, but 🛛	works for loading a	ddresses			tØ		5	8×808084
		c58593 addi x11,x11								t1		6	9×808686
		d00613 addi ×12,×0,				with 13. This is th				t2 50		7	0x000000
	0x08480818 8x84 0x08480814 8x88	000893 addi x17,x0,			hat determines whi y issue the call	ch system call we a	re ca			50 51		9	0x00000x0
		0000/3ecall 000513addi x10.x0,		iall #actuall ia0,0 #The exi	y issue the call	returning is 0				51		10	0×0000000
		d00893 addi x10,x0,				what system call w	e are			a0 a1		11	0x100100
	x08480828 8x80			all	e need to indicate	what system catt v	e are			a2		12	8×808080
· ·										a3		13	0x00000x0
										84		14	8x808080
										a5		15	8×808080
								✓ Data	✓ Text	a6		16	9×808080
									//	a7		17	8×808080
										\$2		18	090909x9
					Data Segment					\$3		19	9×9999999
ddress	Valu	e (+0) Valu	e (+4) Valu			e (+10) Valu	e (+14)	Value (+18) V	alue (+1c)	s4 s5		20	000000x0
	0×10010000	Øx6c6c6548	0x61572061	0x21646c72	6x8080808a	05060608X8	8×80808086		0×08080808	55		22	0300003X9
	0×10010020	8×80808080	8×80808080	8×80808080	8×80808080	8×80808080	8×80808080	0x080808080	0×08080808	s0 s7		22	0x0000000
	0×10010040	09090909x9	05050505×6	08080808x8	08080808x8	8×80808080	8×80808080	60505050X0	60808080×0	° 58		24	0x0000000
	0×10010060	030080803×6	030080803×6	05080808×6	6x66666666	08060608x8	8×80808080	6x08080808	60808080×0	59		25	8×808080
	0×10010080	020080809x9	020080809x9	8x80808080	8x80808080	6×60808080	8×80808086	0x08080808	0x08080808	s10		26	8×898989
	0×100100a0	09090909×9	08080808×8	08080808×8	08080808×9	08080808×9	8×80808086	60808080×0	80808080×0	s11		27	0x000000
	0×100100c0	030080809×6	08080808X9	08080808×0	08080808x9	08080808×8	8×80808086		60808080×0	t3		28	8x808080
	0x100100e0	0x00000000	080080808x9	05080808×8	05060606×8	8×80808080	6×60808086	6×080808080	80808080×0	t4		29	8×808080
	0×10010100	08080808×0	08080808×0	05050508×8	08080808×0	6×60808080	6×60808086	60505050×0	6×08080808	t5		30	090000x6
	0×10010120	09090909×9	08080809×9	05050508×6	05050508×9	05050505×6	8×80808086	60505080×0	66808080×0	t6		31	8x808080
		9×99999999	6x60808080	05080808×0	050606080	0x00000000	8×80808086	0x080808080	0×08080808	pc			0x804080
	0×10010140			8×80808080	0x00000000	8×80808080	8×80808080	80808080×0	00808080x0				
		03000009x9	05050505×6	000000000									
	0×10010140 0×10010160				Hexadecimal Addres	ses 🔽 Hexadecim	al Values 🗌 A	SCII					
	0×10010140 0×10010160		000000000		Hexadecimal Addres	ses 🗹 Hexadecim	al Values 🗌 A	SCII	h.				
	0×10010140 0×10010160		000000000	(.data) 🔿 🗸	•	ses 🗹 Hexadecim	al Values 🗌 A	SCII	ĥ				
	8×10010140 8×10010160 8×10010100	•••••	000000000	(.data) 🔿 🗸	Hexadecimal Addres	ses 🗹 Hexadecim	al Values 🗌 A	SCII	h				
	8×10010140 8×10010160 8×10010100		000000000	(.data) 🔿 🗸	•	ses 🕜 Hexadecim	al Values 🗌 A	SCII	li.				
	9x10919140 9x10919160 	•••••	→ 0x10010000	(.data) 🔿 🗸	•	ses 🕜 Hexadecim	al Values 🗌 A	SCII					
Clear	9x10919140 9x10919160 	es ranzance remnang	→ 0x10010000	(.data) 🔿 🗸	•	ses 🗹 Hexadecim	al Values 🛛 A	SCII					

- 3. Convert and execute the loop.s file from RARS repo in the test folder.
 - a. Download or cut-paste the raw file (<u>https://raw.githubusercontent.com/TheThirdOne/rars/master/test/loop.s</u>) to use it with RARS. Convert the loop.s program to C program that do the same as loop.s. You can execute the C program from <u>https://repl.it/languages/c</u>. In the loop.s file, check <u>https://github.com/TheThirdOne/rars/wiki/Environment-Calls</u> to understand the ecall. For this one, ecall is just a "return <code>" call in C.
 - b. Assembly and run the loop.s program in RARS. See below screen shot. Check the address, binary code, instructions and source of the assembled code, and also check the register values and memory values (data segment part) of the program execution. After you run the program multiple times, you should run step-by-step, i.e. instruction by instruction and observe the change of values in registers and other locations. During the step-by-step simulation in RARS, do the simulation in your mind of the corresponding C program to understand how high-level language program are actually executed by a computer.

		30 5 C	% 🗊 🕻) 🕙 🗳			Run speed at max						
						Edit Execute					Registers	Floating Point	Control and S
	•			Т	ext Segment				Lai	bels	Name	Number	Value
								Labe		Address 🔺	zero		0 0×00 1 0×00
Program	n Argument	:s:							(global)		sp		2 0x7f
Bkpt A	Address	Code Basic		Source				mair		6×60400000	gp		3 0×10
	0×00400000	0x03c00293 addi	x5,x0,0x0000	303c 5:	li t0, 60					0,000,00000	tp		4 0x00
	0×00400004	0x00000313 addi	x6, x0, 0x0000	3000 6:	li t1, 0				loop.s		t0		5 0x00
		0x00530313 addi			addi t1, t1, 5			loop		0×00400008	t1		6 0x00
	0x0040000c	0xfff28293 addi	x5,x5,0xffff	ffff 9:	addi t0, t0, -	-1		failu		0×00400018	t2		7 0×00
		0xfe531ce3 bne >			bne t1, t0, lo			succ	ess	0×00400024	s0		8 0x00
		0x00031863 bne >			bne t1, zero,	success					s1		9 0×00
		0x00000513 addi			li a0, 0						a0		10 0×00
		0x05d00893 addi			li a7, 93						a1		11 0×00
		0x00000073 ecall		15:	ecall						a2		12 0×00
		0x02a00513 addi			li a0, 42						a3		13 0×00
		0x05d00893 addi		3005d 19:	li a7, 93						a4		14 0x00
	0x0040002c	0x00000073 ecal1		20:	ecall						a5		15 0×00
									Data	✓ Text	a6		16 Ø×00
										//	a7		17 0×00
											s2		18 0×00
						Data Segment					s3		19 0×00
ddress		Value (+0)	Value (+4)	Va		ie (+c) Value ((+10) Valu	e (+14) Valu	e (+18)	Value (+1c)	s4		20 0×00
uuress	0×1001000			×00000000	0×00000000	0x00000000	0x00000000	0x00000000	0×00000000	0x00000000	s5		21 0×00
	0x1001002			×000000000	0×000000000	0×000000000	0x00000000	0×000000000	0×000000000	0×000000000	s6		22 0x00
	0x1001002			x000000000	0×00000000	0x00000000	0x000000000		0×000000000	0×00000000	s7		23 0x00
											° s8		
		0 0,00000					0-00000000	0x00000000					24 0×00
	0x1001006		909 9	×00000000	0×00000000	0×00000000	0x00000000	0×00000000	0×00000000	0×00000000	s9		25 Øx00
	0×1001006 0×1001008	0 0x00000	000 0 000 0	×00000000 ×00000000	0×00000000 0×00000000	0x000000000 0x000000000	0×00000000	0×00000000 0×00000000	0×000000000000000000000000000000000000	0×00000000 0×00000000	s9 s10		25 0x00 26 0x00
	0x1001006 0x1001008 0x100100a	0 0×00000 0 0×00000	000 0 000 0 000 0	×80868688 ×80868688 ×80868688	0×00000000 0×00000000 0×00000000	0×00008080 0×00008080 0×00008080	0x00000000 0x00000000	0×00000000 0×00000000 0×00000000	0×00000000 0×00000000 0×00000000	0×00000000 0×00000000 0×00000000	s9 s10 s11		25 0x00 26 0x00 27 0x00
	0x1001006 0x1001008 0x100100a 0x100100a	0 0x00000 0 0x00000 0 0x00000	000 0 000 0 000 0 000 0	×00000000 ×00000000 ×00000000 ×00000000	0×00000000 0×00000000 0×00000000 0×000000	0x00000000 0x00000000 0x00000000 0x000000	0×080808080 0×080808080 0×080808080	0×00000000 0×00000000 0×00000000 0×000000	0×00000000 0×00000000 0×00000000 0×000000	0×00000000 0×00000000 0×00000000 0×000000	s9 s10 s11 t3		25 0x00 26 0x00 27 0x00 28 0x00
	0x1001006 0x1001008 0x100100a 0x100100c 0x100100c	0 0×00000 0 0×00000 0 0×00000 0 0×00000	000 0 000 0 000 0 000 0 000 0	×00000000 ×00000000 ×00000000 ×00000000	0×00000000 0×00000000 0×00000000 0×000000	0×00000000 0×00000000 0×00000000 0×000000	0x00000000 0x00000000 0x00000000 0x000000	0×0000000 0×0000000 0×0000000 0×0000000 0×000000	0×00000000 0×00000000 0×00000000 0×000000	0×0000000 0×0000000 0×0000000 0×0000000 0×000000	s9 s10 s11 t3 t4		25 0x00 26 0x00 27 0x00 28 0x00 29 0x00
	0x1001006 0x1001008 0x100100a 0x100100c 0x100100c 0x100100e	0 0x00000 0 0x00000 0 0x00000 0 0x00000 0 0x00000 0 0x00000	000 0 000 0 000 0 000 0 000 0 000 0	×00000000 ×00000000 ×000000000 ×00000000	0×00000000 0×00000000 0×00000000 0×000000	0x0000000 0x0000000 0x0000000 0x0000000 0x000000	0x00000000 0x00000000 0x00000000 0x000000	0×00000000 0×00000000 0×00000000 0×000000	0x00000000 0x00000000 0x00000000 0x000000	0×0000000 0×0000000 0×0000000 0×0000000 0×000000	s9 s10 s11 t3 t4 t5		25 0x00 26 0x00 27 0x00 28 0x00 29 0x00 30 0x00
	0x1001006 0x1001008 0x1001008 0x1001000 0x1001000 0x1001010 0x1001010	0 0×00000 0 0×00000 0 0×00000 0 0×00000 0 0×00000 0 0×00000 0 0×00000 0 0×00000 0 0×00000 0 0×00000	000 0 000 0 000 0 000 0 000 0 000 0 000 0	x0000000 x00000000 x00000000 x00000000 x000000	0×00000000 0×00000000 0×00000000 0×000000	0×00000000 0×0000000 0×0000000 0×0000000	0x00000000 0x00000000 0x00000000 0x000000	0x00000000 0x00000000 0x00000000 0x000000	0×00000000 0×00000000 0×00000000 0×000000	0×0000000 0×0000000 0×0000000 0×0000000 0×000000	s9 s10 s11 t3 t4 t5 t6		25 0x00 26 0x00 27 0x00 28 0x00 29 0x00 30 0x00 31 0x00
	0x1001006 0x1001008 0x100100a 0x100100c 0x100100c 0x1001010 0x1001010 0x1001012 0x1001014	0 0x00000	000 0 000 0 000 0 000 0 000 0 000 0 000 0	×00000000 ×00000000 ×000000000 ×00000000	0×0000000 0×0000000 0×0000000 0×00000000	0×0000000 0×0000000 0×0000000 0×0000000 0×000000	0x0000000 0x0000000 0x0000000 0x0000000 0x000000	8×06000000 0×0600000 0×0600000 0×0600000 0×0600000 0×0600000 0×0600000 0×06000000	8×96969996 9×9699999 9×9699999 9×96969999 8×96969999 9×96969999 9×96969999 8×96969999	0x0000000 0x0000000 0x00000000 0x0000000	s9 s10 s11 t3 t4 t5		25 0x00 26 0x00 27 0x00 28 0x00 29 0x00 30 0x00
	0x1001006 0x1001008 0x1001008 0x1001000 0x1001000 0x1001010 0x1001010	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	000 0 000 0 000 0 000 0 000 0 000 0 000 0 000 0 000 0 000 0 000 0 000 0 000 0	x0000000 x00000000 x00000000 x00000000 x000000	0×00000000 0×00000000 0×00000000 0×000000	0×00000000 0×0000000 0×0000000 0×0000000	0x00000000 0x00000000 0x00000000 0x000000	0x00000000 0x00000000 0x00000000 0x000000	0×00000000 0×00000000 0×00000000 0×000000	0×0000000 0×0000000 0×0000000 0×0000000 0×000000	s9 s10 s11 t3 t4 t5 t6		25 0x00 26 0x00 27 0x00 28 0x00 29 0x00 30 0x00 31 0x00
	0x1001006 0x1001008 0x1001008 0x1001000 0x100100c 0x100100c 0x10010102 0x1001012 0x1001012 0x1001012 0x1001015 0x1001015 0x1001015	0 0x80808 0x80	000 0 000 0 000000	x00000000 x00000000 x00000000 x00000000	0 × 00000000 0 × 0000000 0 × 00000000 0 × 00000000	0x0000000 0x0000000 0x0000000 0x0000000 0x000000	0x00006000 0x00006000 0x00000000 0x00000000	0.x0000000 0.x0000000 0.x0000000 0.x0000000 0.x0000000 0.x00000000	8×90000900 8×90000900 0×90000000 8×90000000 8×90000000 8×90000000 0×00000000 0×00000000 8×00000000 0×00000000 8×00000000	0x0000000 0x0000000 0x0000000 0x0000000 0x000000	s9 s10 s11 t3 t4 t5 t6		25 0x00 26 0x00 27 0x00 28 0x00 29 0x00 30 0x00 31 0x00
Clear	0x1001096 0x1001008 0x1001008 0x1001002 0x100100c 0x100100c 0x1001010 0x1001012 0x1001015 0x1001015 0x1001015 0x1001015 0x1001015 0x1001015 0x100105 0x100105 0x100105 0x100105 0x100105 0x100105 0x100105 0x100105 0x100105 0x100105 0x100105 0x100105 0x100105 0x100105 0x100105 0x100105 0x100105 0x100105 0x100105 0x100105 0x1005	0 0 xx0000 0 0x0000 0 0x0000 0 0x0000 0 0x00000 0 0x00000 0 0x00000 0 0x00000	000 0 000 0 000000	x00000000 x00000000 x00000000 x00000000	0 × 00000000 0 × 0000000 0 × 00000000 0 × 00000000	0:x0000000 0:x0000000 0:x0000000 0:x00000000	0x00006000 0x00006000 0x00000000 0x00000000	0.x0000000 0.x0000000 0.x0000000 0.x0000000 0.x0000000 0.x00000000	8×90000900 8×90000900 0×90000000 8×90000000 8×90000000 8×90000000 0×00000000 0×00000000 8×00000000 0×00000000 8×00000000	0x0000000 0x0000000 0x0000000 0x0000000 0x000000	s9 s10 s11 t3 t4 t5 t6		25 26 27 28 29 30

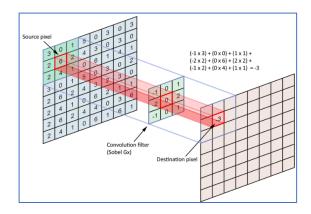
- 4. Implement a program to accumulate integer numbers from 1 to 100 using both C and RISC-V assembly, and simulate the assembly program execution using RARS.
 - a. Write a main program in C from <u>https://repl.it/languages/c</u> to accumulate integers from 1 to 100 together using a for loop. The program should print the result (using C printf) and also retune the value of the accumulation (return). Execute the program from the browser and make sure it produces the expected output (5050 I believe).
 - b. Using the loop.s as starting point, program 1-100 integer accumulation using RISC-V assembly. While the instructions we learned during the class should be sufficient to do the work, you can check RARS supported instructions (<u>https://github.com/TheThirdOne/rars/wiki/Supported-Instructions</u>) and use them. To print the result and return the result, your program should make environment call PrintInt, check <u>https://github.com/TheThirdOne/rars/wiki/Environment-Calls</u>.
- 5. Implement a program to find the average of 100 integers that are randomly generated, using both C and RISC-V assembly, and simulate the assembly program execution using RARS.

- a. The C program MUST follow these steps: 1) declare an int array of 100 elements, and use a for loop to generate 100 integers and store them in the array; 2) use another for loop to accumulate those numbers by reading them from the array and adding up to a variable; 3) calculate the average by dividing the accumulated sum with 100, and 3) print the average and return the average. Your program should NOT do the number generation and accumulation in one loop. Write C program from <u>https://repl.it/languages/c</u>. The functions for generating random numbers can be found from Lab 03 1-D stencil code.
- b. Converting the C program to RISC-V assembly and simulate its execution using RARS. To use array in assembly code, your code needs to reserve space in the data section. Check the memory.s file in RARS repo (<u>https://github.com/TheThirdOne/rars/blob/master/test/memory.s</u>) for using .space to

reserve memory for an array identified by a symbol, and how to use la instruction to load the memory address (first element of the array) to a register.

- 6. Implement a program to do the simple 1-D stencil of 100 integers that are randomly generated, using both C and RISC-V assembly, and simulate the assembly program execution using RARS.
 - a. For C program, convert the 1-D stencil program from lab_03 to use array reference (B[i]) to access array element instead of using pointers. The C program follows these steps: 1) declare two arrays, each has 100 elements, 2) use a for loop to randomly generate 100 integers and store them in one array; 2) use another for loop to do the 1-D stencil and store the result in the other array; Write a main program in C from https://repl.it/languages/c.
 - b. Converting the C program to RISC-V assembly and simulate the program execution using RARS.
- 7. Implement a RISC-V assembly program for image convolution. Image convolution perform 3x3 filtering of each pixel of the source image and the result is written to the dest image. It is the operation widely used for many image processing algorithms, and convolution is one of the kernels of convolution neural network (CNN) for deep learning. The algorithm can be described using the following C code and figure. C array are stored in row-major and make sure you program calculates address of array elements (e.g. source[i][j]) correctly using what we studied during the class. Implement the program using RISC-V assembly and executes it on RARS.

```
int M = 16. N = 16:
int filter[3][3] = \{-1, 0, 1, -2, 0, 2, -1, 0, 1\};
int source[M][N], dest[M][N];
int i, j;
for (i=0; i<M; i++)
  for (j=0; j<N; j++)
     source [i][i] = rand(); // init data with random
/* convolution kernel */
for (i=1; i<M-1; i++)
  for (j=1; j<N-1; j++) {
     int tmp = 0;
     for (fi=0: fi<3: fi++)
        for (fj=0; fj<3; fj++) {
           tmp += source[i+fi-1][j+fj-1]*filter[fi][fj];
        }
      dest[i][j] = tmp;
  }
```



For submission: please submit a single PDF file that shows the execution screenshot of each program in RARS and C terminal required by this lab, and also submit all the source code (both C and assembly code) as well.