Department of Computer Science
University of North Carolina at Charlotte

ITSC 3181: Introduction to Computer Architecture, Spring 2020
Final Exam

Name: Landon Leigh
Student #: 801072367

Important Notice:

- This is a *open-book*, *open-note* exam. But no electronic devices (laptop, calculator, smart phone, PDA, etc) or internet access are allowed to use other than using the computer to type in your solution.
- Bitwise representation has a subscript "`two`", e.g.
  `1111 1101 1011 1001 0111 1010 0011 0001`$_{two}$
- Hexadecimal representation starts with "`0x`", e.g., `0x20202020`.
- Without specific prefix and subscript indications, the integers shown in this exam are base 10 (decimal) values.

- Print your name and student number in the boxes above and print your name at the top of every page.
- Please write your answers on the front of the exam pages. You can use the backs of the pages as scratch paper. Let us know if you need more paper.
- Read the entire question before writing anything. Make sure you understand what the questions are asking. If you give a beautiful answer to the wrong question, you'll get no credit. If any question is unclear, please ask us for clarification.
- Don't spend too much time on any single problem. If you get stuck, move on to something else and come back later.
- Write something down for every problem. Don't panic and erase large chunks of work. Even if you think it's nonsense, it may be worth partial credit.

For compiling C to RISC-V assembly code, instructions that you might use include:

add rd, rs1, rs2	# rd = rs1 + rs2
addi rd, rs1, #immediate	# rd = rs1 + #immediate
sub rd, rs1, rs2	# rd = rs1 – rs2
slli rd, rs1, #immediate	# shift left logic rs1 by #immediate number # of bits and store results in rd
lw rd, #offset(rs1)	# load a word from memory at address rs1+#offset to rd
sw rs2, #offset(rs1)	# store a word from rs2 to memory at address rs1 + #offset
beq rs1, rs2, #label	# if rs1 == rs2, branch to the instruction labeled as #label
bne rs1, rs2, #label	# if rs1 != rs2, branch to the instruction labeled as #label
bge rs1, rs2, #label	# if rs1 >= rs2, branch to the instruction labeled as #label.
ble rs1, rs2, #label	# if rs1<=rs2, branch to the instruction labeled as #label
bgt rs1, rs2, #label	# if rs1 > rs2, branch to the instruction labeled as #label.

Registers are represented as x0, x1, …, x31, and x0 always contains 0.

To reference a[i] in load and store instruction, you can use **a(xi)** to specify the offset(base) part of the instruction in which **xi** is the register for storing **i**. For example, for reading a[i] if **i** is in register **x10**, you can use "lw x6, **a(x10)**".

1. **(5 %)** Translate the following high-level code to RISC-V assembly. The code is for the inner loop of bubble sort algorithm. A is an integer array with N elements. Assume variable N is stored in x6, and use x10 for variable i. tmp should be stored in x12 if you need to use it.

```
int i, N;
int A[N];
for (i=0; i!=N-2; i++) {
        if (A[i] > A[i+1]) { //swap A[i] and A[i+1]
                int tmp = A[i];
                A[i] = A[i+1];
                A[i+1] = tmp;
        }
}
```

```
        # Initialize registers for variable i
        li x10, 0
        lw x6, a(x10)
        add x7, x5, -2
        add x4, x10, -1

        # branch label and check
loop: bneq x10, x7, exit

        # load A[i] and A[i+1]
        Lw x8, A(x7)
        Lw x9, A(x4)
```

```
            # Compare and swap
            bgt x12, x9, then
            sw x8, x12
            beq x0, x0, stop
    then:   sw x9, x8

            # Increment loop index i and jump to the beginning of the loop
            addi x10, x10, 1
            beq x0, x0, loop
            # loop exit
    exit:
```

2. **(5%)** We use a processor with CPIs for the following classes of instructions to execute the following C-code. 1) approximate how many instructions are to be executed for each of the class), and 2) calculate the total number of cycles for each class and overall total cycles for all classes, and 3) calculate the percentage of total cycles by each class with respect to the overall total cycles. **Addition used for calculating memory effective address for load and store instructions should NOT be considered. But loop count increment is considered as addition.**

Instruction class	CPI
add	2
load/store	8
branch	4

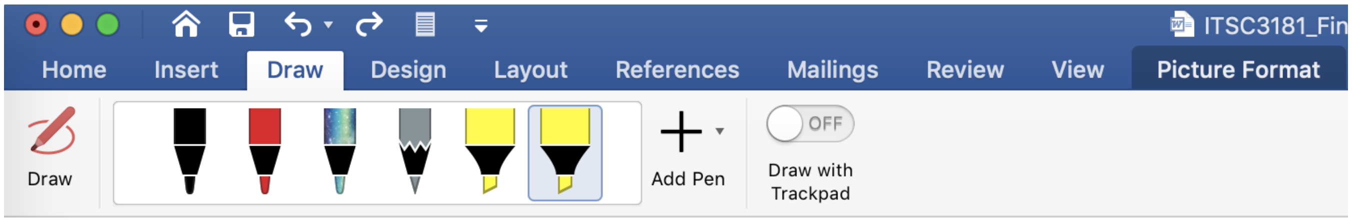
```
int N = 1001;
int A[N]
int i,
for (i=1; i < N-1; i++) {//1000 iterations
   A[i] = A[i-1] + A[i] + A[i+1];
}
```

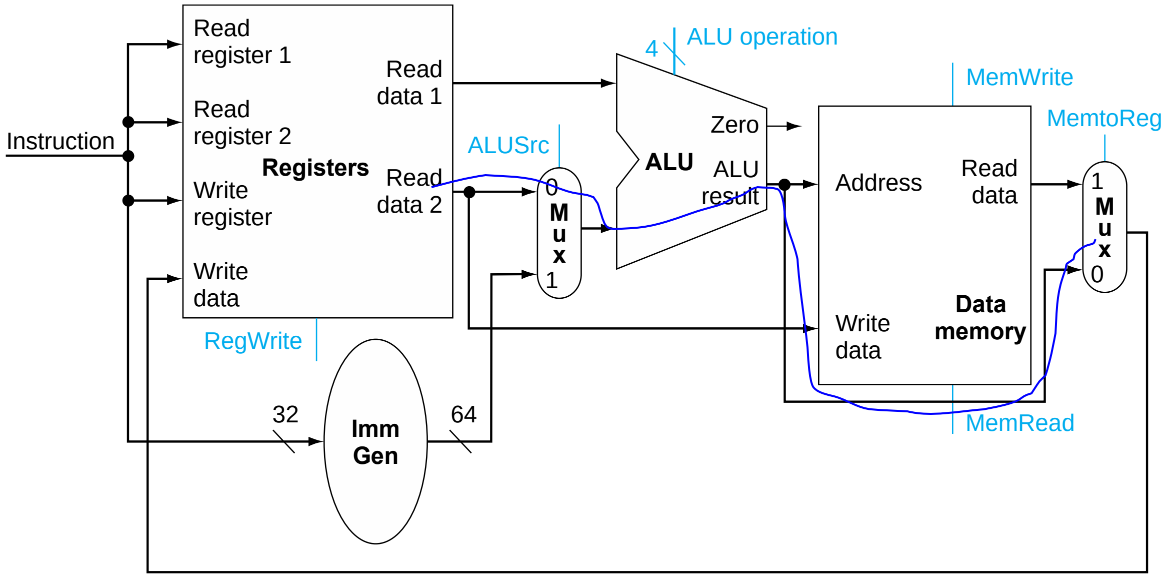
Instruction class	CPI	# instructions x10^3	Total Cycles x10^3	Cycles Percentage
add	2	4000	8000	16.67%
load/store	8	4000	32000	66.67%
branch	4	2000	8000	16.67%

Overall Total cycles = (8000 + 32000 + 8000) * 10^3 = 48000 * 10^3

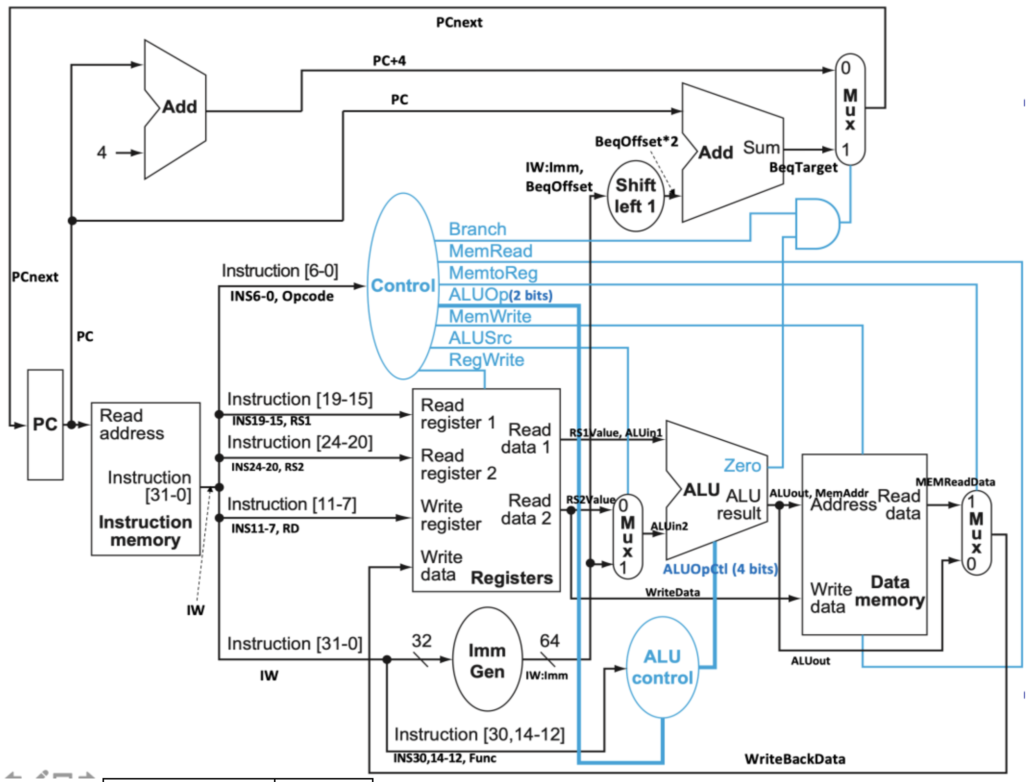
3. **(5%)** Consider two different implementations of an instruction set architecture. P1 has a clock rate of 4.0 GHz and CPIs of 2, 8, and 3 for ALU, load/store and control transfer instructions. P2 has a clock rate of 3.8 GHz and CPIs of 1, 10, and 2 for the three classes of instructions. Given a program with a dynamic instruction count of 1.0E6 instructions divided into classes as follows: 40% class ALU, 30% class load/store, 30% class control transfer instructions. What is the global CPI for each implementation? Which implementation is faster?
CPI P1 = 0.4*2 + 0.3*8 + 0.3*3 = 4.1
CPI P2 = 0.4*1 + 0.3*10 + 0.3*2 = 4
P2 has less CPI so it is faster

4. **(5%)** The following diagram shows the data path and control signals of a standard 5-stage CPU design for load, store, and arithmetic/logic instructions. Highlight the data path used for store instruction. Store is in the format of “sw Rs2, Rs1, imm”. The operation is to perform Mem[Reg[Rs1] + Imm] = Reg[Rs2]. To draw from MS Word, using the draw feature of the Word application, see below screen shot of the menu:





5. **(15%)** Given the following CPU diagram that you use for your Homework 4, for this question, you will be asked in the same way as the question 2 of HW4 using the following diagram. Given instructions to be executed on the CPU, fill in a table with values for selected datapath for each instruction when being executed on CPU.



Register #	value
0	0
1	10
2	16
3	32
4	24
5	12
6	0

Memory Address	value
16	100
20	110
24	114
30	118
34	120
38	140

Your answer should be in the following table. (Instructions are independently executed. Datapath whose values are not used for the instruction should be marked X, even if they may have valid values.). branch target address is calculated by PC+#label*2

Instr Address		INS19-15, RS1	INS24-20, RS2	INS11-7, RD	RS1Value, ALUin1	RS2Value	ALUin2	ALUout	MEMout	WriteBack Data	PCnext
4	add x3, x2, x1	2	1	3	16	10	10	26	X	26	8
12	**sub x1, x5, x4**	**11**	**7**	**15**	**4**	**1**	**1**	**3**	**x**	**3**	**8**
16	**addi x6, x2, 14**	**0**	**x**	**3**	**0**	**X**	**1**	**1**	**x**	**1**	**28**
20	**lw x2, 14(x4)**	**5**	**X**	**6**	**24**	**X**	**-4**	**20**	**x**	**0**	**32**
32	**sw x5, -8(x3)**	**10**	**9**	**X**	**24**	**120**	**44**	**44**	**x**	**X**	**38**
48	**beq x6, x0, 4**	**3**	**4**	**X**	**2**	**140**	**12**	**140**	**x**	**X**	**16**

6. **(10%)** Based on the same diagram as Question 5, fill in the following table by setting the values (0, or 1) of each control signal. For the ALU operation, you only need to write down the English word for the operation, e.g. add, sub, etc

	RegWrite	ALUSrc	ALU Operation	MemRead	MemWrite	MemtoReg
Add	1	1	add	0	0	0
Addi	1	1	add	0	0	0
Or	0	1	or	1	1	1
Load	1	0	add	1	0	1
Store	0	0	add	0	1	0
beq	0	1	sub	0	0	0

7. **(20%)** Pipeline execution and RAW (Read-After-Write) data hazard. For the following RISC-style instruction sequence, (15%)

```
addi x2, x0, 128
lw   x5, 0(x2)
add  x6, x3, x5
add  x7, x4, x5
add  x8, x7, x6
sw   x8, 32(x2)
add  x2, x2, 4
```

a) Fill in the following table about RAW dependency between two consecutive instructions and also mark those RAW dependencies that are load-use. You do not need to use all the rows of the table.

Instruction that writes the register	Instruction that reads the register	The register
addi x2, x0, 128	**lw x5, 0(x2)**	**X2**
lw x5, 0(x2)	**Add x6, x3, x5**	**X5**
Add x6, x3, x5	**Add x8, x7, x6**	**X6**
Add x7, x4, x5	**Add x8, x7, x6**	**X7**
Sw x8, 32(x2)		**X8**

b) Draw the 5-stage pipeline execution using stage labels (IF, ID, EXE, MEM, and WB). CPU has no any forwarding, register files can be read/write in the same cycle, and instruction memory and data memory are separated. Use X to indicate a stalled cycle.

c) Draw the 5-stage pipeline execution using stage labels on the same CPU but with fully forwarding.

d) For the CPU with fully data forwarding, rearrange instructions to eliminate the stall(s) from load-use hazard and then draw the 5-stage pipeline execution of the rearranged instruction sequence.

	Cycles																		
b) No Forwarding	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
`addi x2, x0, 128`	IF	ID	EX	ME	WE														
`lw x5, 0(x2)`		IF	ID	EX	ME	WE													
`add x6, x3, x5`					IF	ID	EX	ME	WE										
`add x7, x4, x5`						IF	ID	EX	ME	WE									
`add x8, x7, x6`							IF	ID	EX	ME	WE								
`sw x8, 32(x2)`										IF	ID	EX	ME	WE					
`add x2, x2, 4`											IF	ID	EX	ME	WE				
														IF	ID	EX	ME	WE	
c) Full Forwarding																			
`addi x2, x0, 128`	IF	ID	EX	ME	WE														
`lw x5, 0(x2)`		IF	ID	EX	ME	WE													
`add x6, x3, x5`				IF	ID	EX	ME	WE											
`add x7, x4, x5`					IF	ID	EX	ME	WE										
`add x8, x7, x6`						IF	ID	EX	ME	WE									
`sw x8, 32(x2)`							IF	ID	EX	ME	WE								
`add x2, x2, 4`								IF	ID	EX	ME	WE							
d) Rescheduling																			

8. **(10%)** A direct-mapped cache has 128 (2^7) entries of cache lines. Each cache line contains tag, valid bit and an 8-word data block. Each word is 4 bytes.

a) Calculate the number of bits of each field in a memory address that are used to access the cache and the value of each field for memory address 0x0FABC8A5. Provide your answer in the following table:

Fields	**Tag**	**Block index**	**Word offset**	**Byte offset**
Number of bits	5	4	7	28

0x0FABC8A5	0x05	4	7	28

b) How many bytes (round to integer) are required for each cache line in this cache?

9. **(15%)** For the following C code of summing up two two-dimensional array data. Arrays are row-major stored. int has 4 bytes: (8 points)

```
sum = 0;
int A[N*N]; int B[N*N];
for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    sum += A[i*N+j] + B[j*N+i];
return sum;
```

a) List the variable references that exhibit temporal locality and those that exhibit spatial locality.

Temporal Locality: i, j, A[i*N+j], B[j*N+i]

Spatial Locality: sum

b) Assuming we have a cache that has two 4-word cache blocks (each cache block can store 4 integers). The two blocks are used for array A and B only, one for A and one for B. For the execution of the code for the inner loop assuming N=1000, fill in the following table array element access, cache Miss/Hit, stride, # memory access, # misses and miss rate. A and B are perfectly 16-byte memory-aligned starting from the first element, and cache blocks are initially empty. The table is exactly the same as the table in Lab13/14, but transposed from the way it was shown.

	Array	**Load for A**	**Load for B**	
j=0	Element	A[0]	B[0]	
	Cache H/M	M	M	
j=1	Element	A[N]	B[1]	
	Cache H/M	M	H	
j=2	Element	A[2N]	B[2]	
	Cache H/M	M	H	
j=3	Element	A[3N]	B[3]	
	Cache H/M	M	H	

	Element	A[4N]	B[4]	
j=4	Cache H/M	M	H	
	Element	A[5N]	B[5]	
j=5	Cache H/M	M	H	
…	…	…	…	
j=1000	…	…	…	
	Stride	**N**	**1**	**Total**
	# Accesses	**64**	**64**	**128**
	# Misses	**64**	**8**	**72**
	Miss Rate	**100%**	**13%**	
Summary	**Notes**			

10. **(10%)** CPU execution time and CPI is computed as follows when considering memory access stalls. (8 points)

Assume we have a computer where the clocks per instruction (CPI) is 1.0 when all memory accesses hit in the cache. The only data accesses are loads and stores, and these total 40% of the instructions. If the miss penalty is 200 clock cycles and the miss rate is 2% for I-Cache and 5% for D-cache, how much faster would the computer be if all instructions were cache hits? Memory stall cycles include the stall cycles for both instruction memory access and data memory access.

I-mem: 0.02 * 200 = 4
D-mem: 0.4 * 0.05 * 100 = 4
Actual CPI = 1 + 4 + 4 = 9
9/1 = 9 times faster
4 + 4 / 9 = 8/9 = 88.89%