Your Name (please print clearly) ____

This exam will be conducted according to the Georgia Tech Honor Code. I pledge to neither give nor receive unauthorized assistance on this exam and to abide by all provisions of the Honor Code.

Signed ____

1	2	3	4	total
30	24	21	25	100

Instructions: This is a closed book, closed note exam. Calculators are not permitted.

Read each question over before you start to work. If you need to make any assumptions, state them. If you have a question, raise your hand; do not leave your seat. The meaning of each question should be clear, but if something does not make any sense to you, please ask for clarification.

Please work the exam in pencil and do not separate the pages of the exam. If you run out of room, please continue on the back of the previous page. For maximum credit, show your work.

Good Luck!

https://clipartoons.com/wp-content/uploads/2016/01/best-skateboard-clipart-1.png

Problem 1 (2 parts, 30 points) Loops

Part A (12 points) Suppose the following declaration is given of an array `Timeline[10]` of alternating year, city integers, as in Homework 2. Assume the initial value shown below is just an example; it can be initialized with a different set of years/cities, under the following constraints. The years range from 1986 to 2015, inclusive, and each year indicates when a move was made to a new city. The total number of moves in each timeline is exactly five. Write a program that computes the total number of moves (`NumMoves`) that were made in the 1990's. ***For maximum credit, declare and initialize any necessary variables.***

```
int Timeline[10] = {1987, 2, 1993, 9, ... 2014, 7}; // given
int NumMoves;
```

Part B (18 points) Write MIPS code for the fragment in Part A. **Store the `NumMoves` computed in register $2**. *For maximum credit use a minimum number of instructions.*

Label	Instruction	Comment
	.data	#
TimeLine:	.word 1986, 4, 1988, …, 2	# given Timeline
	.text	#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#

Problem 2 (2 parts, 24 points) **Conditionals: Nested if-then-else**

We have learned that there are many conditional structures, such as if-then-else, compound predicate assignments, ternary decision statements, and switch statements. For the following MIPS code, assume that $2, $3, and $4 are assigned to integers x, y and z respectively.

```
start:      addi  $1, $0, 10
            beq   $1, $2, L10
            addi  $1, $0, 20
            beq   $1, $2, L20
            j     Ld
L10:        add   $3, $3, $4
            j     end
L20:        addi  $3, $0, 0
Ld:         addi  $4, $0, 0
end
```

Part A (8 points) Draw the control flow graph for the MIPS code shown.

Part B (16 points) Write the C code that corresponds to the above MIPS code with (possibly nested) if-then-else statements, and **without** any if-then-else statements (hint: switch will work).

With if-then-else statements	Without if-then-else statements

Problem 3 (3 parts, 21 points) **Assembly Programming**

Part A (8 points) The following buggy code fragment was supposed to sum up all the (unsigned) hexadecimal digits in the word stored at XLoc and to put the sum in register $5. Sometimes it gives the correct answer and sometimes it doesn't. Give an example initial value (in hex) stored at XLoc that will result in the correct sum and give an example initial value (in hex) where the resulting sum is incorrect.

Example initial value that gives **correct** sum:	**0x**
Example initial value that gives **incorrect** sum:	**0x**

Label	Instruction	Comment
	.data	#
XLoc:	.word 0x__________________	#
	.text	#
	addi $7, $0, 0	# $7: byte offset
	addi $5, $0, 0	# $5: running sum
	addi $4, $0, 4	# $4: constant 4
Loop:	beq $7, $4, ExitLoop	# if $7 == 4, exit the loop
	lbu $3, XLoc($7)	#
	add $5, $5, $3	#
	addi $7, $7, 1	# increment byte offset $7
	j Loop	# loop back
ExitLoop:	...	# stuff after the loop

Part B (8 points) Suppose register $3 has 2 hexadecimal digits in its lower 8 bits (least significant byte) and zeros in the upper 24 bits (upper 3 bytes). Write a MIPS code fragment to sum the two hexadecimal digits and put the result in $5.

Label	Instruction	Comment

Part C (5 points)Write a **single** MIPS instruction that is equivalent to the original fragment. Assume *little endian* byte ordering.

Original:	Equivalent MIPS statement:
lui $4, 0xFF	
lw $3, 1000($0)	
and $3, $3, $4	
srl $3, $3, 16	

Problem 4 (2 parts, 25 points)	**Nonlocal Control Flow**

Part A (12 points) What does the following code fragment print?

```
int i, x;
int A[10] = {99, 33, 44, 22, 56, 78, 1, 5, 9, 88};

for(i=0; i<10; i++){
  if(i & 3)             \\ bitwise AND
     continue;
  x = A[i];
  printf("x = %d\n", x);
}
```

Fill in the blanks to rewrite the code above to produce the equivalent behavior without using `continue`.

```
int i;
int A[] = {99, 33, 44, 22, 56, 78, 1, 5, 9, 88};

for(__________; __________; __________){
  x = A[i];
  printf("x = %d\n", x);
}
```

Part B (13 points) Answer the three questions below about the following C fragment.

```
int i, j, k, count;
j = k = count = 0;

for (i=0; i < 8; i++){                        \\ outer loop
   if (i % 2) continue;
   while (j < 8){                             \\ middle loop
          while (k < 8){                      \\ inner loop
              if (k % 2) break;
               k++;
          }
          count++;
          j++;
   }
}
printf("%d\n", count);
```

How many times is `break` executed?
How many times is `continue` executed?
What is printed?

MIPS Instruction Set (core)

instruction	***example***	***meaning***
arithmetic		
add	add $1,$2,$3	$1 = $2 + $3
subtract	sub $1,$2,$3	$1 = $2 - $3
add immediate	addi $1,$2,100	$1 = $2 + 100
add unsigned	addu $1,$2,$3	$1 = $2 + $3
subtract unsigned	subu $1,$2,$3	$1 = $2 - $3
add immediate unsigned	addiu $1,$2,100	$1 = $2 + 100
set if less than	slt $1, $2, $3	if ($2 < $3), $1 = 1 else $1 = 0
set if less than immediate	slti $1, $2, 100	if ($2 < 100), $1 = 1 else $1 = 0
set if less than unsigned	sltu $1, $2, $3	if ($2 < $3), $1 = 1 else $1 = 0
set if < immediate unsigned	sltui $1, $2, 100	if ($2 < 100), $1 = 1 else $1 = 0
multiply	mult $2,$3	Hi, Lo = $2 * $3, 64-bit signed product
multiply unsigned	multu $2,$3	Hi, Lo = $2 * $3, 64-bit unsigned product
divide	div $2,$3	Lo = $2 / $3, Hi = $2 mod $3
divide unsigned	divu $2,$3	Lo = $2 / $3, Hi = $2 mod $3, unsigned
transfer		
move from Hi	mfhi $1	$1 = Hi
move from Lo	mflo $1	$1 = Lo
load upper immediate	lui $1,100	$1 = 100 x 2^16
logic		
and	and $1,$2,$3	$1 = $2 & $3
or	or $1,$2,$3	$1 = $2 | $3
and immediate	andi $1,$2,100	$1 = $2 & 100
or immediate	ori $1,$2,100	$1 = $2 | 100
nor	nor $1,$2,$3	$1 = not($2 | $3)
xor	xor $1, $2, $3	$1 = $2 ⊕ $3
xor immediate	xori $1, $2, 255	$1 = $2 ⊕ 255
shift		
shift left logical	sll $1,$2,5	$1 = $2 << 5 (logical)
shift left logical variable	sllv $1,$2,$3	$1 = $2 << $3 (logical), variable shift amt
shift right logical	srl $1,$2,5	$1 = $2 >> 5 (logical)
shift right logical variable	srlv $1,$2,$3	$1 = $2 >> $3 (logical), variable shift amt
shift right arithmetic	sra $1,$2,5	$1 = $2 >> 5 (arithmetic)
shift right arithmetic variable	srav $1,$2,$3	$1 = $2 >> $3 (arithmetic), variable shift amt
memory		
load word	lw $1, 1000($2)	$1 = memory [$2+1000]
store word	sw $1, 1000($2)	memory [$2+1000] = $1
load byte	lb $1, 1002($2)	$1 = memory[$2+1002] in least sig. byte
load byte unsigned	lbu $1, 1002($2)	$1 = memory[$2+1002] in least sig. byte
store byte	sb $1, 1002($2)	memory[$2+1002] = $1 (byte modified only)
branch		
branch if equal	beq $1,$2,100	if ($1 = $2), PC = PC + 4 + (100*4)
branch if not equal	bne $1,$2,100	if ($1 ≠ $2), PC = PC + 4 + (100*4)
jump		
jump	j 10000	PC = 10000*4
jump register	jr $31	PC = $31
jump and link	jal 10000	$31 = PC + 4; PC = 10000*4