Your Name (please print clearly) _____

Your Student ID (9 digit)_____

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	total
45	25	30	100

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

Please work the exam in dark pencil or pen and do not separate the pages of the exam. Note that this exam is double sided. If a line or box is provided for your answer, please write only the answer there. Additional explanations can be placed outside the specified area. If you run out of room, please mark on the problem that you will continue on the pages at the end of the exam. Also please identify the problem that is being continued, and draw a box around it.

Read each question over before you start to work.

If you have a question, raise your hand and I will come to you; do not leave your seat.

For maximum credit, show your work.

Good Luck!

Problem 1 (3 parts, 45 points) **Loops, Memory Access, and Compound Predicates**

Part 1A (30 points) Suppose there are 20 bounding boxes stored in memory. Each bounding box is specified by four positive integers: `Tx, Ty, Bx, By`, where `(Tx, Ty)` is the top left coordinate of the box and `(Bx, By)` is the bottom right coordinate, so `Tx<Bx` and `Ty<By`. (This is the same representation used in Homework 1.) The following C code fragment finds the `Tx` position of the leftmost box.

```
int BBs[] = {10, 15, 40, 50,  // 1st BBox: (10,15) (40,50)
             22, 15, 36, 20,  // 2nd BBox: (22,15) (36,20)
             ...,             // . . . 17 more BBoxes
             45, 10, 60, 25}; // 20th Bbox: (45,10) (60,25)
int i, minX = BBs[0];
for (i=4; i<80; i=i+4){
  if (BBs[i] < minX)
     minX = Bbs[i];
}
```

Suppose the 20 bounding boxes are stored in memory beginning at the address labeled `BBs.` Write MIPS code that is equivalent to the C fragment above. Store the `Tx` of the leftmost bounding box in the memory location labeled `minX`. *For maximum credit, include comments and consider the address offset between successive bounding boxes.* (Note: there may be more blank lines provided than you need.)

Label	Instruction	Comment
	.data # Tx Ty Bx By	
BBs:	**.word** 10, 15, 40, 50	**# 1st BBox: (10,15) (40,50)**
	.word 22, 15, 36, 20	**# 2nd BBox: (22,15) (36,20)**
	.word ...	**# . . . 17 more BBoxes**
	.word 45, 10, 60, 25	**# 20th Bbox: (45,10) (60,25)**
minX:	**.alloc** 1	
	.text	

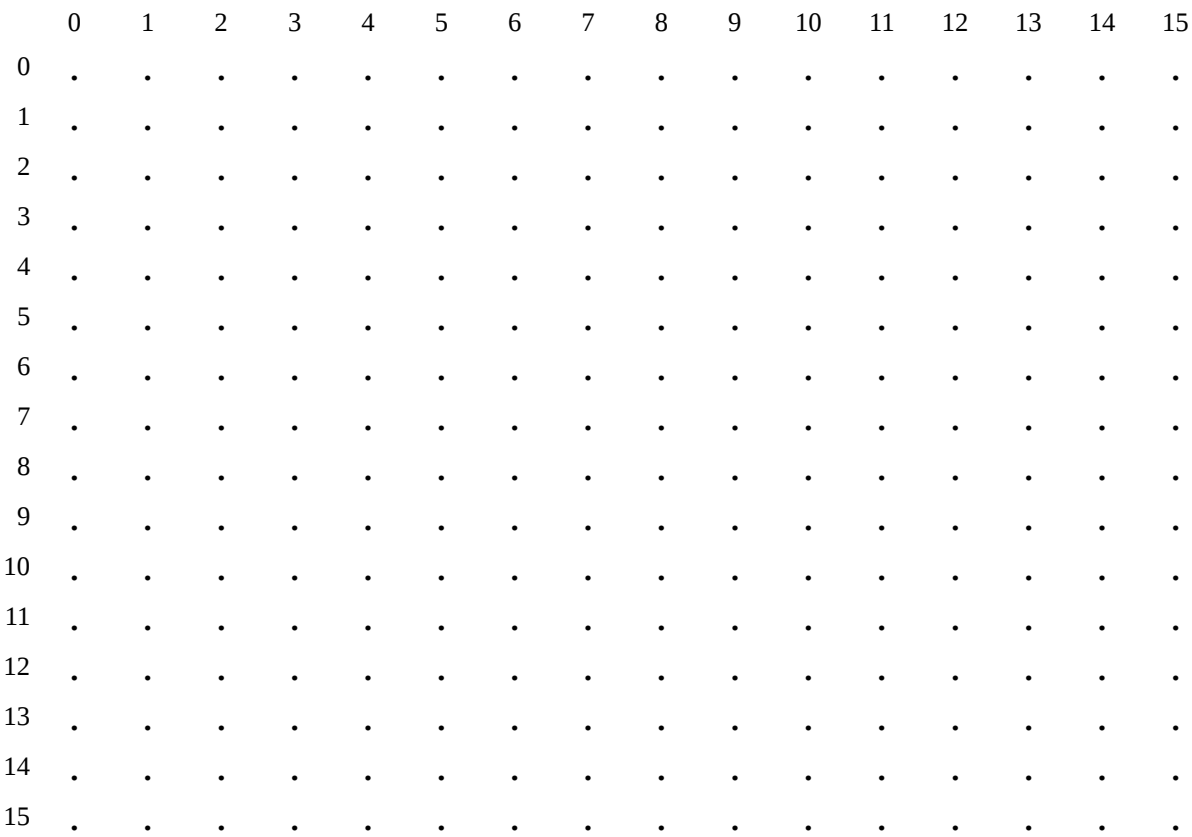
Part 1B Suppose a bounding box, specified as in Part A, is stored in registers as follows: `$1` holds `Tx`, `$2` holds `Ty`, `$3` holds `Bx`, and `$4` holds `By`. Register `$5` holds variable `Limit` which is the width-1 (and height-1) of a square image and `$6` holds variable `D`. Consider the following MIPS code:

```
        beq $1, $0, R
        beq $2, $0, R
        sub $7, $3, $5
        beq $7, $0, R
        sub $7, $4, $5
        beq $7, $0, R
        addi $6, $0, 1
        j    End
   R: addi $6, $0, 0
End: ...
```

Part 1B.1 (10 points) What C statement does this MIPS code implement? Express your answer as a single assignment statement using a compound predicate expression involving variables `Tx`, `Ty`, `Bx`, `By`, and `Limit`.

D = _____;

Part 1B.2 (5 points) Connect dots to draw an example of a bounding box for which D is assigned 1. The rows and columns give the x and y axes of the image. Assume `Limit` is 15.



Problem 2 (4 parts, 25 points) **MIPS Instructions**

Part 2A (6 points) For the following MIPS code fragment, write an equivalent MIPS code fragment in the fewest possible instructions. If the original is already using the fewest number of instructions, write "no change".

Original fragment:	*Equivalent fragment in minimal number of MIPS instructions:*
`addi $1, $0, 1`	
`sll  $1, $1, 7`	
`mult $1, $2`	
`mflo $3`	

Part 2B (6 points) For the following MIPS code fragment, write an equivalent MIPS code fragment in the fewest possible instructions. If the original is already using the fewest number of instructions, write "no change".

Original fragment:	*Equivalent fragment in minimal number of MIPS instructions:*
`addi $12, $2, 1`	
`slt  $13, $3, $12`	
`bne  $13, $0, End`	

Part 2C (5 points) For the following MIPS code fragment, write an equivalent MIPS code fragment in the fewest possible instructions. If the original is already using the fewest number of instructions, write "no change".

Original fragment:	*Equivalent fragment in minimal number of MIPS instructions:*
`sub $7, $3, $5`	
`beq $7, $0, R`	

Part 2D (8 points) Assuming *little* endian byte ordering, values are in registers $4 and $5 after this MIPS code fragment executes? Express your answers as **8-digit hexadecimal** numbers.

```
.data
Input: .word 0xA26B0C8D
.text
       addi $3, $0, Input
       lb   $4, 1($3)
       lb   $5, 3($3)
```

`# $4: 0x`
`# $5: 0x`

Problem 3 (1 part, 30 points) **Sparse Vectors and Nonlocal Control Flow**

Suppose the variable `TL` declared below holds a sparse vector representing a timeline similar to the ones used in Homework 2, except that the timeline is longer (it holds 50 moves) and there are 64 possible cities. The even indices each hold a duration (an integer number of years) while the odd indices each hold a city code (an integer between 0 and 63).

Write a C code fragment to determine whether all cities in the timeline `TL` are unique (i.e., that there are no duplicate city codes in the timeline), and if so, set variable `unique` to 1, otherwise set it to 0. Assume adjacent moves in the timeline have different city codes. **For full credit, declare and initialize any necessary variables and end the loop as early as possible** – if a duplicate city code is found, stop looping and report answer using the given `printf` statement.

```
int TL[100] = {4, 45, 3, 23, …, 1 19};
unsigned unique;
```

```
printf("%s cities are unique.\n", (unique ? "All" : "Not all"));
```

Extra work space:

If you are continuing a problem here, please identify the problem that is being continued.

Extra work space:

If you are continuing a problem here, please identify the problem that is being continued.

Extra work space:

If you are continuing a problem here, please identify the problem that is being continued.

Do not write on this page.

Tear off this last page to use MIPS ISA (on other side) as reference.

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 + sign_extend(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 + sign_extend(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 & zero_extend(100)
or immediate	ori $1,$2,100	$1 = $2 | zero_extend(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 ⊕ zero_extend(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