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`	
	`lw  $2, BBs($0)`	**# $2: minX = BBs[0]**
	`addi $1, $0, 16`	**# $1: i, offset into BBox**
Loop:	`slti $3, $1, 320`	**# is i<80?**
	`beq  $3, $0, Exit`	**# if not, Exit loop**
	`lw  $4, BBs($1)`	**# $4 = BBs[i]**
	`slti $5, $4, $2`	**# is BBs[i]<minX?**
	`beq  $5, $0, Skip`	**# if not, skip updating minX**
	`addi $2, $4, 0`	**# update minX = BBs[i]**
Skip:	`addi $1, $1, 16`	**# i = i+4**
	`j Loop`	**# keep looping**
Exit:	`sw $2, minX($0)`	**# stores $2 at minX**

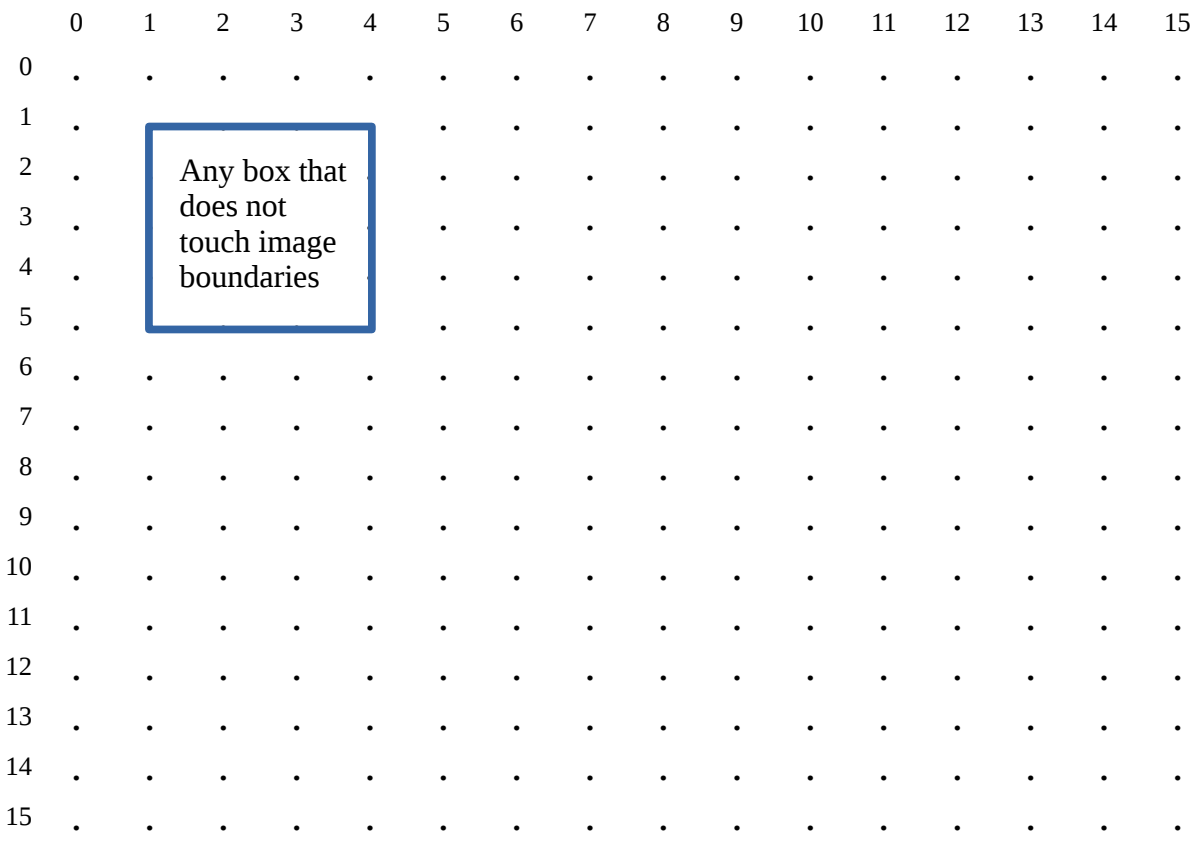
Part 1B Suppose a bounding box, specified as in Part A, is stored in registers as follows: `$1` holds `Tx`, `$2` holds `Ty`, `$3` holds `Bx`, and `$4` holds `By`. Register `$5` holds variable `Limit` which is the width-1 (and height-1) of a square image and `$6` holds variable `D`. Consider the following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

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 $3, $2, 7`
`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, $2, $3`
`slt  $13, $3, $12`	`beq $13, $0, End`
`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 $3, $5, R`
`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: 0x0000000C`
`# $5: 0xFFFFFFA2`

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=1;
int i, j;
for (i=1; i<98; i+=2){
  for(j=i+4; j<100; j += 2){
    if (TL[i] == TL[j]){
      unique = 0;
      break;
    }
  }
  if (!unique) break;
}
printf("%s cities are unique.\n", (unique ? "All" : "Not all"));
```