Instructions: This is a closed book, closed note exam. Calculators are not permitted. If you have a question, raise your hand and I will come to you. Please work the exam in pencil and do not separate the pages of the exam. For maximum credit, show your work.
Good Luck!

Your Name (***please print***) ____

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	5	6	total
20	30	20	22	18	40	150

Problem 1 (20 points) **Compilation**

Perform at least **five** standard compiler optimizations on the following C code fragment by writing the optimized version (in C) to the right. *Assume* ***f*** *is a pure function that returns an integer with no side effects to other data structures.*

```
int foo(int g, int h) {
  int p = 1, y, j;
  int x = 0, z = 24;
  for (j=100; j > 0; j--) {        ⇨
    x += f(j+g+h);
    y = x/z + g*h;
    p *= f(y) - (j+g+h)/128;
  }
  return (p);
}
```

Briefly describe which standard compiler optimizations you applied:

1.

2.

3.

4.

5.

Problem 2 (2 parts, 30 points) **Packed Pixel Data**

Suppose an image is stored in memory as an array of pixels. As in Homework 2, each pixel is represented as a triple of 8-bit red, green, and blue color components, packed in the lower 24 bits of a 32-bit word, as shown here:

31			0
unused	red component	green component	blue component
31 24	23 16	15 8	7 0

Part A (20 points) Write a MIPS code fragment that reads in the red, green, and blue components of the i^{th} pixel of the image, finds the maximum of these color components, and stores it in register $11. Assume $1 holds *i*, which could be any integer 0, 1, 2,...N-1, where N is the number of pixels in the image. The image is stored in memory starting at base address labeled `Image`. *Modify only registers $1, $2, $3, $4, $5, and $11.*

Label	Instruction	Comment
`MaxPixel:`		`# Compute address of i_th`
		`# pixel and put it into $1`
		`# Read R, G, B components of`
		`# i_th pixel into $2, $3, $4`
		`# Find the max(R, G, B) and`
		`# put it in $11.`

Part B (10 points) Suppose we have an image processing application that reads in the pixels in an image array `Image` and unpacks the color components as shown in the C fragment below. Complete the fragment (by filling in the blank) so that it repacks the color components into the same pixel location but with the red and blue components swapped.

```
int  i, Blue, Green, Red;
for (i=0; i<ImageSize; i++){
   /* unpack current color */
   Color = Image[i];
   Blue = Color & 0xFF;
   Green = (Color >> 8) & 0xFF;
   Red = (Color >> 16) & 0xFF;
   /* Repack the pixel color components with Red and Blue swapped. */


   Image[i] = ______________________________________________________;
  }
```

Problem 3 (2 parts, 20 points) **Reverse Engineering MIPS Assembly and C**

Part A (10 points) The following MIPS code implements a three-dimensional array access.

```
sll $1, $4, 11
sll $2, $5, 7
add $1, $1, $2
add $1, $1, $6
sll $1, $1, 2
add $1, $1, $3
lw  $2, 0($1)
```

This implements the C code below. The base address of the array **`Video`** is stored in `$3`. Variables **`Frame`**, **`Row`**, **`Col`**, and **`Pixel`** reside in `$4`, `$5`, `$6`, and `$2` respectively. Fill in the array declaration below. Assume a 32-bit operating system.

```
int     Video[8192][________][_______]; /* array declaration */

Pixel = Video[Frame][Row][Col]; /* array access */
```

Part B (10 points) Consider the following MIPS code fragment. If $1, $2, $3, and $4 hold variables A, B, C, and D, respectively, what is the equivalent 1-line C statement using compound predicates that this computes? Hint: draw the control flow graph.

Label	Instruction
	`addi $4, $0, 0`
	`beq  $1, $0, TestC`
	`bne  $2, $0, TestC`
	`j End`
`TestC:`	`bne  $3, $0, End`
	`addi $4, $0, 1`
`End:`	`...`

Answer: `D = ______________________________________________________;`

Problem 4 (3 parts, 22 points) **Garbage Collection**

Below is a snapshot of heap storage. Values that are pointers are denoted with a "$". The heap pointer is $6168. The heap has been allocated contiguously beginning at $6000, with no gaps between objects.

addr	value	addr	value	addr	value	addr	value	addr	value	addr	value
6000	8	6032	12	6064	0	6096	16	6128	12	6160	0
6004	33	6036	28	6068	4	6100	6172	6132	$6120	6164	0
6008	$6100	6040	$6120	6072	$6100	6104	16	6136	9	6168	0
6012	16	6044	$6080	6076	8	6108	5	6140	6072	6172	0
6016	80	6048	16	6080	24	6112	148	6144	20	6176	0
6020	8	6052	$6072	6084	$6132	6116	8	6148	6046	6180	0
6024	25	6056	$6080	6088	4	6120	32	6152	8	6184	0
6028	$6004	6060	0	6092	80	6124	$6080	6156	26	6188	0

Part A (10 points) Suppose register $3 holds the address $6004 and the stack holds a local variable whose value is the memory address $6052. No other registers or static variables currently hold heap memory addresses. List the addresses of all objects in the heap that are *not* garbage.

Addresses of

Non-Garbage Objects: ______________________________

Part B (3 points) If a reference counting garbage collection strategy is being used, what would be the reference count of the object at address $6100?

Reference count of object at $6100 = ______________________________

Part C (9 points) If the local variable whose value is the address $6052 is popped from the stack, which addresses from Part A will be reclaimed by each of the following strategies? If none, write "none."

Reference Counting:	
Mark and Sweep:	
Old-New Space (copying):	

Problem 5 (2 parts, 18 points) **Linked Lists and Pointers**

Consider a singly linked list whose elements are `Car` structs defined as follows:

```
typedef struct Car {
  int Year;
  int Tag;
  struct Car *Next;
} Car;
```

The global `KnownCars`, which is declared and initialized as follows, holds the head of the list.

```
Car *KnownCars = Null;
```

Part A (10 points) Suppose the list is sorted in order of increasing `Tag` numbers. Complete the C function `Lookup_Car` below that efficiently searches the list for a `Car` that has the `TagNum` given as input. It should return a pointer to the matching `Car` if `TagNum` is found or return `Null` otherwise.

```
Car *Lookup_Car(int TagNum) {
   Car      *ThisCar;















}
```

Part B (8 points) Consider the procedure `Lookup_Car`. In what region of memory is each of the following allocated? (Put a checkmark in the column of the correct memory region containing each.)

	Static	Heap	Stack	OS
KnownCars (pointer to head of list)				
ThisCar (pointer to a Car)				
the Car object pointed to by ThisCar				
TagNum (integer input to Lookup_Car)				

Problem 6 (2 parts, 40 points) Activation Frames

The function `Bar` (below left) calls function `Foo` after completing `code block 1`. Write MIPS assembly code that properly calls `Foo`. Include all instructions between `code block 1` and `code block 2`. Symbolically label all required stack entries and give their values if they are known (below right).

```
int Bar() {
   int         A = 25;
   int         B[] = {2, 4};

   (code block 1)

   B[0] = Foo(A, &A, B);

   (code block 2)

}
```

Bar's FP 9900	XXX	XXX
9896	A	25
9892	B[1]	4
SP 9888	B[0]	2
9884		
9880		
9876		
9872		
9868		
9864		
9860		
9856		

label	instruction	comment
		# allocate activation frame
		# preserve bookkeeping info
		# push inputs
	jal Foo	# call Foo
		# restore bookkeeping info
		# read return value
		# store return value in B[0]
		# deallocate activation frame

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