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	total
24	40	36	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 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 (2 parts, 24 points) **Storage Allocation, Strings, and Pointers**

Part A (16 points) **Assuming a 64-bit system with 64-bit memory interface and 64-bit addresses**, answer the following addressing questions. Assume all alignment restrictions imposed by the hardware are obeyed, and the compiler does not add additional alignment restrictions. Note: `int` and `float` are 4 bytes, and `double` is 8 bytes. For each part below, fill in the value of each expression given that the expression in the comment is true. You may find it helpful to sketch memory allocation including slack for each part. Assume variables are allocated in **global memory** in the order they are declared. **Please only write numbers in each answer box.**

Part A1 (4 pts).

`int i; // &i == 1000`
`char c[2];`
`double x`
	`&c[1]`	

	`&x`	

Part A2 (8 pts).

`char *s; // &s == 1000`
`char d = 'T';`
`struct {`
`    char c;`
`    double y;`
`    int j;`
`    float z;`
`} thing;`
	`&d`	

	`&thing`	
	`&thing.y`	
	`sizeof(thing)`	

Part A3 (4 pts).

`int m; // &m == 1000`
`float *q;`
`int *p = &m;`
	`&q`	

	`p+1`	

Part B (8 points) Fill in what is printed after the following C fragment executes?

```
char  *foo(char *s,char *t){
   char *z = s;
   while (*t){
   if (*t == 't' || *t == 'T')
      t++;
   else
      *s++ = *t++;
   }
   *s = *t;
   return z;
}
char a[25];
char b[] = “This cart tis there!”;
printf(”%s\n”,foo(a,b)); //only write exactly what is printed
```

What is printed?	

Problem 2 (4 parts, 40 points) **Accessing Inputs, Locals, Arrays**

Part A (16 points) Consider the following C code on a 32-bit machine:

```
typedef unsigned char color;
typedef struct {
      color r;
      color g;
      color b;
} pixel;
pixel frame[256][1024];
int i,j;
color c;
…
c = frame[i][j].g // for some i,j  IMPLEMENT THIS LINE
```

Assuming `i,j,frame,` and `&c` are in `$1, $2, $3,` and `$10` respectively, write MIPS code to implement the indicated line. Do not overwrite any given registers, and use additional registers beginning at `$4.` More line are provided than are necessary.

Label	Instruction	Comment

Assuming a 32-bit system, consider the following C fragment:

```
int Rotate (int Pattern[]) {
   int i = 0;
    . . .
       Pattern[i] = Pattern[i+1];      // Part C
    . . .
   return (i);                         // Part D
}
```

Part B (6 points) Write a MIPS code fragment to implement the beginning of Rotate's implementation: set Rotate's frame pointer and allocate its locals.

Label	Instruction	Comment

Part C (12 points) Suppose the input parameter `Pattern` is stored in `Rotate`'s activation frame just above the return value slot (pointed to by the frame pointer $30). Write a MIPS code fragment to implement the line that has the comment "PartC" in the C code above:

```
Pattern[i] = Pattern[i+1];
```

Do not assume that the variable i is already in a register (read it from the stack).

Label	Instruction	Comment

Part D (6 points) Write a MIPS code fragment to store the return value of Rotate, deallocate its locals and return to its caller. In other words, implement the line that has the comment "Part D" in the C code above: `return(i);`

Do not assume that the variable i is already in a register (read it from the stack).

Label	Instruction	Comment

Problem 3 (2 parts, 36 points) **Parameter Passing w/ Activation Frames**

The function `Solver` (below left) calls function `Quad` after completing `code block 1`. Write MIPS code that properly calls `Quad`. Include all instructions between `code block 1` and `code block 2`. Symbolically label all required stack entries at the point just before control is transferred to `Quad` and give their values if they are known (below right).

```
int Quad(int x, int *y, int z[])
{
   . . .
}

int Solver(int a) {
   int b;
   int c[] = {4, 2, -1};

   <code block 1>

   c[2] = Quad(a, &b, c);

   <code block 2>

}
```

...	...	...
9604	a	6
Solver's FP 9600	XXX	XXX
9596	b	
9592	c[2]	-1
9588	c[1]	2
SP 9584	c[0]	4
9580		
9576		
9572		
9568		
9564		
9560		
9556		
9552		

label	instruction	comment
		# Allocate activation frame
		# Preserve bookkeeping info
		# Push inputs
	jal Quad	# Call Quad
		# Restore bookkeeping info
		# Read return value
		# Store return value in c[2]
		# 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 + 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