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
30	30	40		100

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

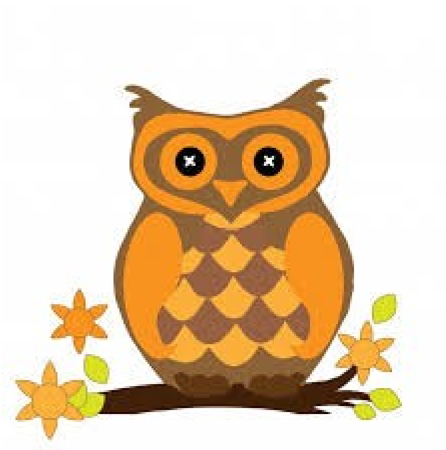
Read each question over before you start to work.

If you 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!



Problem 1 (2 parts, 30 points) **Storage Allocation, Strings, and Pointers**

Part A (20 points) Assuming a **32-bit system with 32-bit memory interface and 32-bit addresses**, show how the following global variables map into static memory. Assume they are allocated starting at address **6000** and are properly aligned. **For each variable, draw a box showing its size and position** in the word memory shown below in which byte addresses increment from left to right. **Label the box with the variable name.** Label each element of an array (e.g., M[0]) and struct (e.g., M.part). 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.

```
typedef struct{
  int x;
  int y;
  double *scale;
} coord;

char f[] = “Buzz”;
char  *h = &f[2];
double z = 4.8;
char   k = ‘!’;
coord  c = {5, 6, &z};
coord *d = &c;
int    e = c.x;
```

6000				
6004				
6008				
6012				
6016				
6020				
6024				
6028				
6032				
6036				
6040				
6044				
6048				
6052				
6056				
6060				

Part B (10 points) What does the following C fragment print?

```
char *S = “twelve+one”;
int  A[] = {2, 3, 5, 4, 9, 8, 6, 0, 1, 7, 10};
int  *V = A;

do {printf(“%c”, *(S + *V++));
   } while(*V != 10);
```

Problem 2 (2 parts, 30 points) **Accessing Arrays and Structs**

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

```
typedef struct{
  int R;
  int G;
  int B;
} Pixel;

Pixel Image[1024][768] = {…};
Pixel *P = Image;

int GetG(int i, int j){
  int Gij = Image[i][j].G;
  return (Gij);
}
```

Part A (10 points) Replace the assignment of `Gij` with a statement that uses only the identifiers `P`, `i`, `j`, and `G` to access the value of `Image[i][j].G`. *Do not use the identifier* `Image` *in your answer*.

`int Gij = ___________________________________;//an expression is ok`

Part B (20 points) Write MIPS code to implement the assignment statement

`int Gij = Image[i][j].G;`

from the code above. Assume that the base address of array **`Image`** is given in **$1**, and the values of variables **i**, and **j** are given in **$2**, and **$3**, respectively. Store the result (**`Gij`**) in register **$4**. (Note: there are more blank lines provided than you need.)

Label	Instruction	Comment

Problem 3 (2 parts, 40 points) **Activation Frames**

Consider the following C code fragment:

```
int Bar() {
   char        x = ‘i’;
   int         y = 1;
   int         z;
   char        Name[] = “Tom”;
   int         Foo(char [], int, char *);

   z           = Foo(Name, y, &x);
   return(z);
}

int Foo(char S[], int n, char *c) {
   int         a = 3;

   if (S[n]) {
      S[n] = *c;
      n++;
   }
   return(n);
}
```

Part A (18 points) Suppose `Bar` has been called so that the state of the stack is as shown below. Describe the state of the stack just before `Foo` deallocates locals and returns to `Bar`. Fill in the unshaded boxes to show `Bar's` and `Foo's` activation frames. Include a symbolic description and the actual value (in decimal) if known. For return addresses, show only the symbolic description; do not include a value. *Label the frame pointer and stack pointer*. Assume a **32-bit system** and maintain word alignment.

address	description	Value
9900	**RA of Bar’s caller**	
9896	**FP of Bar’s caller**	
SP, Bar’s FP 9892	**RV**	
9888		
9884		
9880		
9876		
9872		
9868		
9864		
9860		
9856		
9852		
9848		
9844		
9840		
FP: _________ 9836		
SP: _________ 9832		
9828		

Part B (22 points) Write MIPS code fragments to implement the subroutine `Foo` by following the steps below. *Do not use absolute addresses in your code; instead, access variables relative to the frame pointer.* Assume no parameters are present in registers (i.e., access all parameters from `Foo`'s activation frame). You may not need to use all the blank lines provided.

label	instruction	Comment
Foo:		**# set Foo's FP**
		# allocate space for locals
		# initialize locals

 # if S[n] == 0 branch to End. *Be sure to use load/store byte for values of type char.*

 # otherwise, do Then clauses: S[n] = *c;

 # n++;

 # return(n); (store return value, deallocate locals, and return)

End:		

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 + zero_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