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.

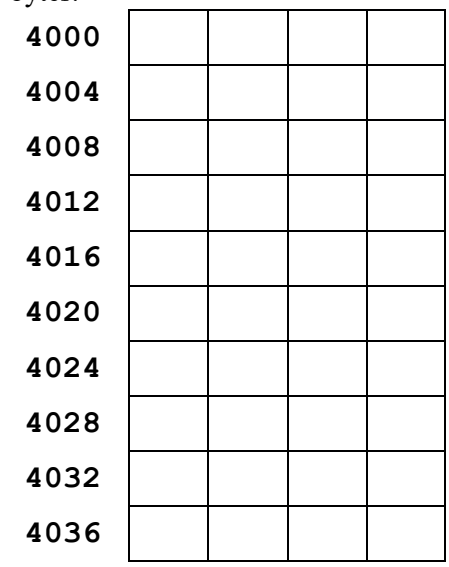
Signed _____

1	2	3	total
30	30	40	100

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

Part A (16 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 4000 and are properly aligned. **For each variable, draw a box showing its size and position** in the 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., N[0]). 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.

```
char    c  = 'a';
char   *p  = &c;
char   N[] = "Now!";
int     x  = 5;
double  d  = 3.23;
double *e  = &d;
```



Part B (14 points) Assuming a 32-bit system, consider the following declaration:

```
int  A[][] = {{2, 6, 10, 14},
              {5, 15, 25, 35},
              {7, 11, 18, 22}};
```

B.1 Array `A` contains an element of value **25** (shown in bold). Write a single C statement that overwrites that element with the value **100**.

_____;

B.2 Write the MIPS code implementation of the following assignment statement in the smallest number of instructions. A pointer to the array **A** is stored in **$1** and variables **`j, i,`** and **`y`** reside in **$2, $3,** and **$4**, respectively. Modify only registers $4 and $5.

```
int y = A[j][i];
```

Label	Instruction	Comment

Problem 2 (3 parts, 30 points) **Accessing structs, pointers, pre/post increment**

Consider the following C code fragment.

```
typedef struct {
   int    A;
   int    B;
   char   C[10];
} struct_t;

int foo() {

   struct_t S[2], *p;
   char *q;

   p = &S[1];
   q = &S[0].C[0];

   S[0].B = 42;
   S[0].C = "yolo"; //copies the string "yolo" to the array S[0].C beginning at C[0]
   p->B = ++(S[0].B);  //This line implemented in assembly for Part C
   *(q++) ='n';
}
```

Part A (15 points) Fill in the value of each expression after the code above is executed.

`S[0].B`	
`S[1].B`	
`S[0].C[0]`	
`S[0].C[4]`	
`(--p)->C[0]`	

Part B (5 points) Assuming a 32-bit system, what is the total space allocated to `foo`'s local variables?

_____bytes

Part C (10 points) Write the assembly code for the line identified above. Hint: Remember they are local variables.

Label	Instruction	Comment

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

Part A (20 points) The function `Bar` (below left) calls function `Area` after completing `code block 1`. Write MIPS assembly code that properly calls `Area`. 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.

```
typedef struct diamond_t {
  int    axis1;
  int    axis2;
} Diamond;

int Area(Diamond *K, int S[], int n){
  int a;
  int m = S[2];
  a = 2 * (K->axis1) * (K->axis2);
  S[1] = m + n;
  return a;
}

int Bar() {
  Diamond D = {5, 10};
  int     V[] = {4, 9, 16};
  int     x = 2;

  <code block 1>

  x = Area(&D, V, x);

  <code block 2>
}
```

Bar's FP	9620	XXX	XXX
	9616	D.axis2	10
	9612	D.axis1	5
	9608	V[2]	16
	9604	V[1]	9
	9600	V[0]	4
SP	9596	x	2
	9592		
	9588		
	9584		
	9580		
	9576		
	9572		
	9568		

label	instruction	comment
		# Allocate activation frame
		# Preserve bookkeeping info
		# Push inputs
	jal Area	# Call Area
		# Restore bookkeeping info
		# Read return value
		# Store return value in x
		# Deallocate activation frame

Part B (20 points) Write MIPS code fragments to implement the subroutine `Area` 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 `Update`'s activation frame). You may not need to use all the blank lines provided.

First, write code to properly set `Area`'s frame pointer and to allocate space for `Area`'s local variables and initialize them if necessary.

label	instruction	Comment
`Area:`		

`# a = 2 * (K->axis1) * (K->axis2);`

label	instruction	Comment

`# S[1] = m + n;`

label	instruction	Comment

`# return a;  (store return value, deallocate locals, and return)`

label	instruction	Comment

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