Your Name (please print clearly) _____

GTID : _____

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
35	40	25	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, 35 points) **Storage Allocation, Strings, and Pointers**

Part A (21 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 (9 points)

`int x; // &x == 1000`
`double A[4][32][10];`
`double *y = A;`
	`&A[2][10][5]`	

	`&y`	
	`y`	

Part A2 (6 points)

`int d = 4; //&d == 1000`
`struct {`
`  char c;`
`  double y;`
`  int j;`
`  float z;`
`} VofS[3];`
	`VofS`	

	`&VofS[1]`	
	`&VofS[1].z`	

Part A3 (6 points)

`float f = 6.3; // &f == 1000`
`float g = 3.14;`
`int  *q;`
`float *p = &f;`
	`&q`	

	`p+1`	
	`p[1]`	

Part B (14 points) Consider the following C fragment and answer the questions below.

```
char SofE[20];
int a, i = 19;
do {                // Part B1
  SofE[i] = 'p';    // Part B1
} while(--i > 1);   // Part B1

char *current = &SofE[2];   // Part B2

for (i = 2; i<20; i++){
  if (*current++ == 'p'){   // Part B2
    printf("%d ", i);
    for (a = 2; a*i < 20; ++a) // HINT for Part B3:
      SofE[a*i] = 'c';         // when i=2: a*i: 4, 6, 8, 10, …
                               // when i=3: a*i: 6, 9, 12, 15, …
                               // when i=4: a*i: 8, 12, 16  and so on
  }
}
```

Part B1 (4 points) How many iterations of the do while loop are executed?

The do while loops this many times:	

Part B2 (5 points) We would like to simplify this code by removing the variable current. If we omit line A , how should we rewrite line B so that it no longer uses current? (To answer, fill in the blank below.) The code fragment should still print the same output.

```
char SofE[20];
int a, i = 19;
do {
  SofE[i] = 'p';
} while(--i > 1);

~~char *current = &SofE[2];~~      // line A

for (i = 2; i<20; i++){

  if (_____){     // line B
    printf("%d ", i);
    for (a = 2; a*i < 20; ++a)
      SofE[a*i] = 'c';
  }
}
```

Part B3 (5 points) What is printed by this C fragment?

What is printed?	

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

Part A (25 points) Consider the following C code on a 32-bit machine:

```
typedef unsigned char color;
typedef struct {
      color r;
      color g;
      color b;
      int   intensity;
} pixel;
pixel frame[256][1024];
int i,j;
int sum=0;
…

// sum the b elements in column j.
for (i=0;i<256;i++) sum = sum + (int) frame[i][j].b; // IMPLEMENT THIS LINE
```

Part A1 (5 points) Evaluate `sizeof(pixel)`._____
Part A2 (5 points) Evaluate `&frame[i+1][j] - &frame[i][j]`. _____

Part A3 (15 points) Assuming `i,j,frame and sum` are in `$1, $2, $3,` and `$10` respectively, write MIPS code to implement the indicated line. Do not overwrite registers $2 and $3. Use additional registers beginning at `$4`. More lines are provided than are necessary. (Hint: use the column stride result from part A2 to simplify your code).

Label	Instruction	Comment
		`# calculate &frame[0][j]`

Part B (15 points)Assuming a 32-bit system, consider the following C fragment:

```
int SetP (int *p) {
   struct {
       int i,j,k;
   } point;
    . . .
       point.k = *p;      // Part B2
    . . .
}
```

Part B1 (5 points) Write a MIPS code fragment to implement the beginning of SetP's implementation: set SetP's frame pointer and allocate its locals.

Label	Instruction	Comment
SetP:		

Part B2 (10 points) Suppose the input parameter `p` is stored in SetP'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 "Part B2" in the C code above:

```
point.k = *p;
```

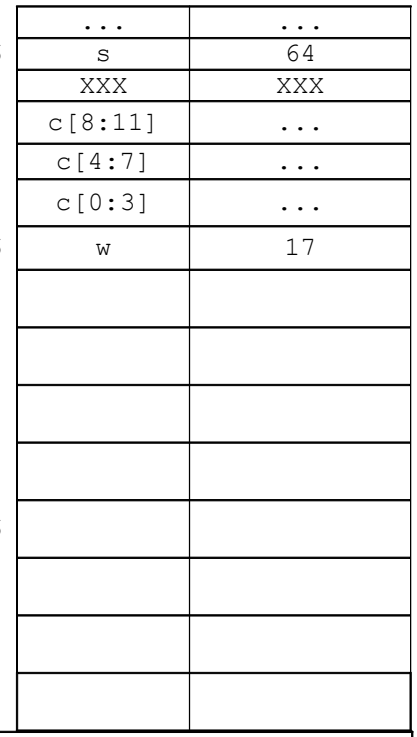
Do not assume that any variable is already in a register (read it from the stack).

Label	Instruction	Comment

Problem 3 (2 parts, 25 points) **Parameter Passing w/ Activation Frames**

The function `Palette` (below left) calls function `DQ` after completing `code block 1`. Write MIPS code that properly calls `DQ`. 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 `DQ` and give their values if they are known (below right).

```
typedef unsigned char color;

int DQ(color P[], int *size, int n){
   . . .
}

int Palette(int s) {
   color c[12];
   int   w = 17;

   <code block 1>

   w = DQ(c, &s, w);

   <code block 2>
}
```



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