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]`	`1008+(2*32*10+10*10+5)*8 =` **`6968`**

	`&y`	`1008+(4*32*10*8) = 1008+10240 =` **`11248`**
	`y`	**`1008`**

Part A2 (6 points)

`int d = 4; //&d == 1000`
`struct {`
`  char c;`
`  double y;`
`  int j;`
`  float z;`
`} VofS[3];`
	`VofS`	**`1008`**

	`&VofS[1]`	**`1032`**
	`&VofS[1].z`	**`1052`**

Part A3 (6 points)

`float f = 6.3; // &f == 1000`
`float g = 3.14;`
`int  *q;`
`float *p = &f;`
	`&q`	**`1008`**

	`p+1`	**`1004`**
	`p[1]`	**`3.14`**

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:	**18**

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 ( SofE[i] == 'p' ){      // 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?	**2 3 5 7 11 13 17 19**

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;89
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)`. **8**

Part A2 (5 points) Evaluate `&frame[i+1][j] - &frame[i][j]`. **8192**

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
	sll $4, $2, 3	# calculate &frame[0][j]
	add $5, $4, $3	
	addi $1, $0, 0	# init i
Loop:	**slti $6, $1, 256**	# is i < 256?
	beq $6, $0, Exit	# if not, exit loop
	lbu $7, 2($5)	# load b member
	addi $10, $10, $7	# update running sum
	addi $5, $5, 8192	# go to next element in column
	addi $1, $1, 1	# i++
	j Loop	# keep looping
Exit:		

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:	**add $30, $29, $0**	**# set FP**
	addi $29, $29, -12	**# make room for local variable (point)**

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
	lw $1, 4($30)	**# load p**
	lw $1, 0($1)	**# dereference P**
	sw $2, -4($30)	**# write it to point.k**

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

The function `Pallete` (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>
}
```

...	...	...
9596	s	64
FP 9592	XXX	XXX
9588	c[8:11]	...
9584	c[4:7]	...
9580	c[0:3]	...
SP 9576	w	17
9572	**RA**	
9568	**FP**	**9592**
9564	**c**	**9580**
9560	**&s**	**9596**
9556	**w**	**17**
9552		
9548		
9544		

label	instruction	comment
	addi $29, $29, -24	# Allocate activation frame
	sw $31, 20($29)	# Preserve bookkeeping info
	sw $30, 16($29)	
	addi $1, $30, -12	# Push inputs: calculate c
	sw $1, 12($29)	# push c
	addi $1, $30, 4	# calculate &s
	sw $1, 8($29)	# push &s
	lw $1, -16($30)	# load w
	sw $1, 4($29)	# push w
	jal DQ	# Call DQ
	lw $31, 20($29)	# Restore bookkeeping info
	lw $30, 16($29)	
	lw $1, 0($29)	# Read return value
	sw $1, -16($30)	# Store return value in w
	addi $29, $29, 24	# Deallocate activation frame