ECE 2035	Programming HW/SW Systems	Spring 2015
3 problems, 6 pages	Exam Two Solutions	4 March 2015

Problem 1 (2 parts, 35 points) **Storage Allocation and Pointers**

Part A (15 points) Assuming a **64-bit system with 64-bit memory interface and 64-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 memory. Label the box with the variable name. Label each element of an array (e.g., M[0]). Note that int and float are still 32-bits.

```
double G;
float  M[] = {1.44, 3.69, 4.64};
double *P = &G;
int    z = 4;
float  *Q = M;
double H;
```

Address	Contents
4000	**G**
4004	
4008	**M[0]**
4012	**M[1]**
4016	**M[2]**
4020	**slack**
4024	**P**
4028	
4032	**z**
4036	**slack**
4040	**Q**
4044	
4048	**H**
4052	

Part B (20 points) For this part, assume a 32-bit system, such as MIPS-32 that we have worked with.

```
int X = 5;
int V = 7;
char S[] = "hello!";
double Z = 7.25;
double *P = &Z;
int *R = &X;
```

Question:	Answer:
How much space (in bytes) is allocated for R?	**4 bytes: |pointer| = |word|**
How much space (in bytes) is allocated for P?	**4 bytes: |pointer| = |word|**
What is printed by this statement? `printf("%d\n", *(R+1));`	**7**
What is printed by this statement? `printf("%c\n", *(S+4));`	**o**
What is printed by this statement? `printf("%f\n", *P);`	**7.25**

Problem 2 (4 parts, 27 points) **Struct and Array Access**

Consider the following declarations.

```
int  A[][] = {{2, 6, 10, 14},
              {5, 15, 25, 35},
              {7, 11, 18, 22}};
typedef struct {
   int height;
   int base;
} triangle;
triangle T;
triangle *P = &T;
```

Part A (3 points) Array `A` contains an element of value **11** (shown in bold). Write a single C statement that overwrites that element with the value **100**.

`A[2][1] = 100;` .

Part B (3 points) Write a C statement that sets the **`base`** field of **`T`** to **16**, using the variable **`T`**.

`T.base = 16;` .

Part C (3 points) Write a C statement that sets the **`height`** field of **`T`** to **20**, using the variable **`P`**.

`P->height = 20;` .

Part D (18 points) Write the MIPS code implementation of the dynamically allocated array access below in the smallest number of instructions. A pointer to the array **`Icons`** (declared below) is stored in `$3`. Variables **`I`**, **`Y`**, **`X`**, and **`Cell`** reside in `$4`, `$5`, `$6`, and `$2` respectively. Modify only registers `$1` and `$2`. Assume a 32-bit operating system.

```
int    Icons[8][12][16];          /* array declaration */
Cell = Icons[I][Y][X];            /* implement this */
```

Label	Instruction	Comment
	`addi $1, $0, 192`	`# Lx*Ly = 192`
	`mult $1, $4`	`# I*Lx*Ly`
	`mflo $1`	`# $1: I*Lx*Ly`
	`sll $2, $5, 4`	`# $2: Y*Lx = Y*16`
	`add $1, $1, $2`	`# $1: I*Lx*Ly+Y*Lx`
	`add $1, $1, $6`	`# $1: I*Lx*Ly+Y*Lx+X`
	`sll $1, $1, 2`	`# scale by 4`
	`add $1, $1, $3`	`# add base`
	`lw $2, 0($1)`	`# read Icons[I][Y][X]`

Problem 3 (2 parts, 38 points) **Activation Frames**

Consider the following C code fragment:

```
int Average(int [], int);

int TrapezoidArea(int H) {
   int         Bases[] = {3, 5};
   int         N = 2;
   int         R;

   R = H * Average(Bases, N);
   return(R);
}

int Average(int A[], int S) {
   int         x, y, avg;
   x = A[0];
   y = A[1];
   avg = (x+y)/S;
   return(avg);
}
```

Part A (18 points) Suppose `TrapezoidArea` has been called with input **`H=10`** so that the state of the stack is as shown below. Describe the state of the stack <u>just before</u> `Average` deallocates locals and returns to `TrapezoidArea`. Fill in the unshaded boxes to show `TrapezoidArea's(TA's)` and `Average's` activation frames. Include a symbolic description and the actual value (in decimal). For return addresses, show only the symbolic description; do not include a value. *Label the frame pointer and stack pointer*.

address	description	Value
9900	**RA of TA's caller**	
9896	**FP of TA's caller**	
9892	**H**	**10**
~~SP,~~TrapezoidArea's FP 9888	**RV**	
9884	**Bases[1]**	**5**
9880	**Bases[0]**	**3**
9876	**N**	**2**
9872	**R**	
9868	**RA**	
9864	**FP**	**9888**
9860	**A**	**9880**
9856	**S**	**2**
Average's FP 9852	**RV**	**4**
9848	**x**	**3**
9844	**y**	**5**
SP 9840	**avg**	**4**
9836		
9832		
9828		

Part B (20 points) Write MIPS code fragments to implement the subroutine `Average` 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 `Average`'s activation frame). You may not need to use all the blank lines provided. (You may assume that intermediate values your code stores in registers stay in the registers from one part to the next unless your code overwrites them.)

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

label	instruction	Comment
`Average:`	`addi $30, $29, 0`	`# set FP base`
	`addi $29, $29, -12`	`# make room for locals (3 words)`

`# x = A[0];`

label	instruction	Comment
	`lw $1, 8($30)`	`# load A (base of array)`
	`lw $2, 0($1)`	`# load A[0]`
	`sw $2, -4($30)`	`# store it in x`

`# y = A[1];`

label	instruction	Comment
	`lw $3, 4($1)`	`# load A[1]`
	`sw $3, -8($30)`	`# store it in y`

`# avg = (x+y)/S;`

label	instruction	Comment
	`add $1, $2, $3`	`# x+y`
	`lw $3, 4($30)`	`# load S`
	`div $1, $3`	`# (x+y)/S`
	`mflo $1`	
	`sw $1, -12($30)`	`# store it in avg`

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

label	instruction	Comment
	`sw $1, 0($30)`	`# store return value`
	`addi $29, $30, 0`	`# deallocate locals`
	`jr $31`	`# return to caller`