ECE 2035 | Programming HW/SW Systems | Spring 2016

3 problems, 6 pages | Exam Two Solutoins | 11 March 2016

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

Part A (16 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 the double 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]). Note: int and float are 32-bits.

```
int      a  = 5;
int      b  = 25;
int     *c  = &b;
double   d  = 9.23;
double  *e  = &d;
char    S[] = "Harry";
char    *f  = &(S[3]);
float    g  = 8.1;
double   z  = 17.25;
```

Address		
4000	**a**	**b**
4008	**c**	
4016	**d**	
4024	**e**	
4032	**S[] (6 bytes)**	**slack (2 bytes)**
4040	**f**	
4048	**g**	**slack (4 bytes)**
4056	**z**	
4064		

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

```
int A[32][16][8] = {...};
int *q = A;
```

B.1 Complete the assignment statement below using only `q` to assign to `x` the value of `A[20][10][2]`.

```
   int x = *(q + 20*16*8+10*8+2 );//an expression is ok
or int x = *(q + 2642 )
```

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 **$3** and variables **j, i,** and **y** reside in **$4, $5,** and **$6**, respectively. Modify only registers $1 and $2.

```
int y = A[0][j][i];
```

Label	Instruction	Comment
	sll $1, $4, 3	# j*8 (= j * Lx)
	add $1, $1, $5	# j*8 + i
	sll $1, $1, 2	# scale by 4
	add $1, $1, $3	# add A
	lw $6, 0($1)	# memory read

Problem 2 (4 parts, 30 points) **Accessing Structs, Activation Frame Allocation**

Consider the following C code fragment.

```
typedef struct {
   int   age;
   float height;   // in meters
   float weight;   // in kilograms
} patient;

float Calc_and_Print_BMI(int A, float H, float W) {
   patient   Pat;
   float     BMI;

   __Pat.age = A;_________________________________// part A

   __Pat.height = H;______________________________// part A

   __Pat.weight = W;______________________________// part A

   BMI = Compute_BMI(&Pat);
   printf("BMI: %f\n", BMI);
   return(BMI);
}

/* BMI = weight divided by height squared (units: kg/m2). */
float Compute_BMI(patient *P) {

   float height_squared = _(P->height)*(P->height)_; // part B


   float BMI = __P->weight__ / height_squared;   //part B

   return(BMI);
}
```

Part A (6 points) Fill in the blanks in `Calc_and_Print_BMI` with statements that assign the inputs `A`, `H`, and `W` to the `age`, `height`, and `weight` fields of `Pat`, respectively.

Part B (6 points) Fill in the blanks in `Compute_BMI` with 1) a statement that squares the `height` of the `patient` pointed to by `P` and 2) a statement that computes the body mass index (`BMI`) of the `patient` pointed to by `P` by dividing the patient's `weight` by `height_squared`.

Part C (12 points) Assuming a 32-bit system, give the total number of bytes allocated for the activation frame of `Calc_and_Print_BMI` and the number of bytes for the activation frame of `Compute_BMI`.

Size of activation frame of `Calc_and_Print_BMI` (in bytes):	**40 bytes**
Size of activation frame of `Compute_BMI` (in bytes):	**24 bytes**

Part D (6 points) What does the following code fragment print?

Code:	**Answer:**

`char Name[] = "Fred";`
`char *p = Name;`
`while (*(++p))`
`   printf("%c\n", *p);`
	r
	e
	d

Problem 3 (1 part, 40 points) **Activation Frames**

The function `Bar` (below left) calls function `Foo` after completing `code block 1`. Write MIPS assembly code that properly calls `Foo`. 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 (below right).

```
int Foo(int Set[], int x, int *y, int z) {
   <body of Foo>
}

int Bar() {
   int         A[] = {40, 45, 50};
   int         B =  10;
   int         i;

   <code block 1>

   A[i] = Foo(A, A[2], &B, 0);

   <code block 2>

}
```

Bar's FP	9620	XXX	XXX
	9616	A[2]	50
	9612	A[1]	45
	9608	A[0]	40
	9604	B	10
~~SP~~	9600	i	assigned in code block 1
	9596	RA	
	9592	FP	9620
	9588	Set	9608
	9584	x	50
	9580	y	9604
	9576	z	0
SP & Foo's FP	9572	RV	

label	instruction	comment
	`addi $29, $29, -28`	`# Allocate activation frame`
	`sw   $31, 24($29)`	`# Preserve bookkeeping info`
	`sw   $30, 20($29)`	
	`addi $1, $30, -12`	`# Push inputs: compute A`
	`sw   $1, 16($29)`	`# push A`
	`lw   $2, 8($1)`	`# read A[2]`
	`sw   $2, 12($29)`	`# push A[2]`
	`addi $2, $30, -16`	`# compute &B`
	`sw   $2, 8($29)`	`# push &B`
	`sw   $0, 4($29)`	`# push 0`
	`jal Foo`	`# Call Foo`
	`lw   $31, 24($29)`	`# Restore bookkeeping info`
	`lw   $30, 20($29)`	
	`lw   $3, 0($29)`	`# Read return value`

	`lw   $2, -20($30)`
	`sll  $2, $2, 2`
	`addi $1, $30, -12`
	`addi $1, $1, $2`
	`sw   $3, 0($1)`
		`# Store return value in A[i]:`
		`#  load i, scale it, and add it`
		`#  to base address of A to`
		`#  compute where to store return`
		`#  value, and store RV there.`

	`addi $29, $29, 28`	`# Deallocate activation frame`