CS61c Spring 2015 Discussion 2 – C Memory Management & MIPS

1 C Memory Management

1. In which memory sections (**C**ODE, S**T**ATIC, **H**EAP, **S**TACK) do the following reside?

```
#define C 2
const int val = 16;
char arr[] = "foo";
void foo(int arg){
    char *str = (char *) malloc (C*val);
    char *ptr = arr;
}
```

arg [C] str [S]

arr [T] *str [H]

val [T] C [T]

2. What is wrong with the C code below?

```
int* ptr = malloc(4 * sizeof(int));
if(extra_large) ptr = malloc(10 * sizeof(int));
return ptr;
```

doesn't cast malloc as int*

memory leak since the original malloc address is lost

3. Write code to prepend (add to the start) to a linked list, and to free/empty the entire list.
`struct ll_node { struct ll_node* next; int value; }`

`free_ll(struct ll_node** list)`	`prepend(struct ll_node** list, int value)`
while (list != NULL){ struct ll_nod	struct ll_node new_head = (struct ll_node*) malloc

Note: `list` *points to the first element of the list, or to* `NULL` *if the list is empty.*

2 MIPS Intro

1. Assume we have an array in memory that contains `int* arr = {1,2,3,4,5,6,0}`. Let the value of `arr` be a multiple of 4 and stored in register `$s0`. What do the following programs do?

a)
```
lw $t0, 12($s0)
add $t1, $t0, $s0
sw $t0, 4($t1)
```
tldr: replaces 3 with 4
stores 4 into $t0
adds 4 to the address
saves 4 over 3

b)
```
addiu $s1, $s0, 27
lh $t0, -3($s1)
```
stores 2 bytes of 0'sinto

c)
```
addiu $s1, $s0, 24
lh $t0$, -3($s1)
```
loads the middle 2 bytes of 6 in $t0, so 0, so 6 is 0000 0000 0000 0110

wrong, actually gives a misalignment error

d)
```
addiu $t0, $0, 12
sw $t0, 6($s0)
```
tries to put 12 into the 6 byte of the array which is half way through two, so it gives a misalgnment error

e)
```
addiu $t0, $0, 8
sw $t0, -4($s0)
```
tries to put 8 into the 4 bytes before the array which is maybe allowed, but a bad idea...

f)
```
addiu $s1, $s0, 10
addiu $t0, $0, 6
sw $t0, 2($s1)
```
loads 6 into 2 bytes past the 10 bytein the a

2. In 1), what other instructions could be used in place of each load/store without alignment errors?

lw, lb, lh; sw, sb, sh

3. What are the instructions to branch to `label:` on each of the following conditions?

`$s0 < $s1`	`$s0 <= $s1`	`$s0 > 1`	`$s0 >= 1`
slt $t0, $s0, $s1 bne $t0, $0, label	slt $t0, $s1, $s0 beq $t0, $0, label	slti $t0, $s0, 2	addi s1, $0, 1 slt $t0, $s0, $s1 beq $t0, $0, label better solution bgtz $s0, label

3 Translating between C and MIPS

Translate between the C and MIPS code. You may want to use the MIPS Green Sheet as a reference. In all of the C examples, we show you how the different variables map to registers – you don't have to worry about the stack or any memory-related issues.

C	MIPS

`// $s0 -> a, $s1 -> b`
`// $s2 -> c, $s3 -> z`
`int a = 4, b = 5, c = 6, z;`
`z = a + b + c + 10;`
	wrong
	addi $t0, $0, 4
	lw $t0, 0($s0)
	addi $t1, $0, 5
	lw $t1, 0($s1)
	addi $t2, $0, 6
	lw $t0, 0($s2)
	add $t0, $t0, $t1
	add $t0, $t0, $t2
	addi $t0, $t0, 10
	lw $t0, 0($s3)
	
	better
	addiu $s0, $0, 4
	addiu $s1, $0, 5
	addiu $s2, $0, 6
	addu $s3, $s0, $s1
	addu $s3, $s3, $s2
	addiu $s3, $s3, 10

`// $s0 -> int * p = intArr;`
`// $s1 -> a;`
`*p = 0;`
`int a = 2;`
`p[1] = p[a] = a;`
	sw $0, 0($s0)
	addiu $s1, $0, 2
	sw $s1, 4($s0)
	sll $t0, $s1, 2
	add $t0, $t0, $s0
	sw $s1, 0($t0)

`// $s0 -> a, $s1 -> b`
 	
`int a = 5, b = 10;`
`if(a + a == b) {`
`  a = 0;`
`} else {`
`  b = a - 1;`
`}`
	addiu $s0, $0, 5
	addiu $s1, $0, 10
	add $t0, $s0, $s0
	bne $t0, $s1, else
	addiu $s0, $0, $0j cont
	
	else:
	addi $s1, $s0, -1
	j cont

// $s0 -> a,// $s1 -> b,int a = 0	`  addiu $s0, $0, 0`
	`  addiu $s1, $0, 1`
	`  addiu $t0, $0, 30`
	`loop:`
	`  beq $s0, $t0, exit`
	`  addu $s1, $s1, $s1`
	`  addiu $s0, $s0, 1`
	`  j loop`
	`exit:`

`// $a0 -> n, $v0 -> sum`
`int sum;`
`for(sum=0;n>0;sum+=n--);`
	add $v0 $0 $0loop:
	beqz $a0, exit
	add $v0, $a0, $v0
	sub $a0, $a0, 1
	j loop
	exit:
	add $v0 $0 $0
	loop:beqz $a0, exit
	add $v0, $a0, $v0
	sub $a0, $a0, 1