University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Spring 2015 Instructors: Krste Asanović, Vladimir Stojanovic 2015-02-26

☹ CS61C MIDTERM 1 ☺

After the exam, indicate on the line above where you fall in the emotion spectrum between "sad" & "smiley"...

Last Name	
First Name	
Student ID Number	
CS61C Login	`cs61c-`
The name of your ***SECTION*** *TA (please circle)*	David | Donggyu | Fred | Jeffrey | Martin Nolan | Sagar | Shreyas | William
Name of the person to your Left	
Name of the person to your Right	
All the work is my own. I had no prior knowledge of the exam contents nor will I share the contents with others in CS61C who have not taken it yet. ***(please sign)***	

Instructions (Read Me!)

- This booklet contains 7 numbered pages including the cover page. **The back of each page is blank and can be used for scratch-work, but will not be looked at for grading.** (i.e. the sides of pages without the printed "SID: _______" header will not even be scanned into gradescope).
- Please turn off all cell phones, smartwatches, and other mobile devices. Remove all hats & headphones. Place your backpacks, laptops and jackets under your seat.
- You have 80 minutes to complete this exam. The exam is closed book; no computers, phones, or calculators are allowed. You may use one handwritten 8.5"x11" page (front and back) of notes in addition to the provided green sheet.
- There may be partial credit for incomplete answers; write as much of the solution as you can. We will deduct points if your solution is far more complicated than necessary. When we provide a blank, please fit your answer within the space provided. "IEC format" refers to the mebi, tebi, etc prefixes.

	Q1	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Total**
Points Possible	10	12	6	12	18	16	16	90
Points Earned								

SID: ________________

Q1: Number Representation (10 points)

1) Convert the following 8-bit two's-complement numbers from hexadecimal to decimal:

0x80 = -128

0xFF = -1

0x0F = 15

2) For two *n*-bit numbers, what is the difference between the largest unsigned number and the largest two's-complement number? In other words, what is **`MAX_UNSIGNED_INT – MAX_SIGNED_INT`**? Write your answer in terms of *n*.

$2^n - 1 - (2^{n-1} - 1) = 2^{n-1}$

3) Fill in the blanks to return the largest positive number a 32-bit two's-complement number can represent.

```
unsigned int max_twos() {
     return ((1 << 31) - 1);
}
```

4) Consider a new type of notation for representing signed numbers, *biased* notation. The formula for obtaining the value from a number written in biased notation is:

value = value_as_unsigned – *b*

Where *b* is a constant called the *bias*. Example with 4 bits and a bias of 4:

```
     0b0011 = 3 – 4 = -1
     0b1110 = 14 – 4 = 10
```

If we wanted an *n*-bit biased system to represent the same range as two's complement numbers, what is the value of *b?*

The most negative number needs to be -2^{n-1}. The most negative number in biased notation is 0 – b so our bias is 2^{n-1}.

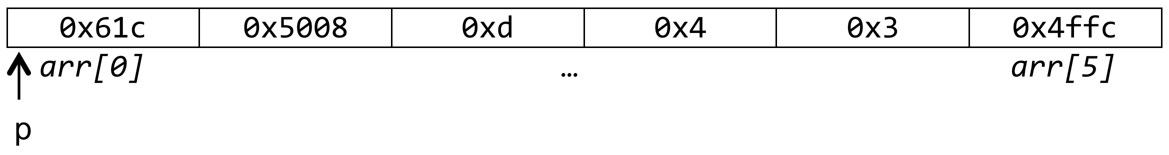
SID: ______________

Q2: Pointers and Memory (12 points)

1) Assume you are given an `int` array `arr`, with a pointer `p` to its beginning:

```
int arr[] = {0x61c, 0x5008, 0xd, 0x4, 0x3, 0x4ffc};
int *p = arr;
```

Suppose `arr` is at location `0x5000` in memory, i.e., the value of `p` if interpreted as an integer is `0x5000`. To visualize this scenario:



Assume that integers and pointers are both 32 bits. What are the values of the following expressions? If an expression may cause an error, write "Error" instead.

a) `*(p+3)` = 0x4

b) `p[4]` = 0x3

c) `*(p+5) + p[3]` = 0x5000

d) `*(int*)(p[1])` = 0xd(13)

e) `*(int*)(*(p+5))` = error(out of bounds)

2) Consider the following code and its output. Fill in the blanks.

```
void foo1( __int*____ a, int n) {
  int i;
  for (i = 0 ; i < n ; i++) {
    (*(a+i)) += 3;
  }
}

void foo2( ___int**____ p) { p++; }

int main() {
  int x = ____4______ ;
  int a[] = {1, 2, 3, 4, 5};
  int *p = &a[1];
  foo1(a, sizeof(a) / sizeof(int));
  foo2(&p);
  printf("%d, %d, %d\n", a[1], *(++p), a[x]);
}
The output of this code is:

_____5______ , _____6______ , 8
```

SID: ________________

Q3: C Memory Model (6 points)

For each of the following functions, answer the questions below in the corresponding box to the right:

1) Does this function return a usable pointer to a string containing **`“asdf”`**?
2) Which area of memory does the returned pointer point to?
3) Does this function leak memory?

You may assume that **`malloc`** calls will always return a non-NULL pointer.

```
char * get_asdf_string_1() {
     char *a = “asdf”;
     return a;
}
```

get_asdf_string_1
1)
2)
3)

```
char * get_asdf_string_2() {
     char a[5];
     a[0]=‘a’;
     a[1]=‘s’;
     a[2]=‘d’;
     a[3]=‘f’;
     a[4]=‘\0’;
     return a;
}
```

get_asdf_string_2
1)
2)
3)

```
char * get_asdf_string_3() {
     char * a  = malloc(sizeof(char) * 5);
     a = “asdf”;
     return a;
}
```

get_asdf_string_3
1)
2)
3)

```
char * g = “asdf”;

char * get_asdf_string_4() {
     return g;
}
```

get_asdf_string_4
1)
2)
3)

Q4: Linked Lists (12 points)

1) Fill out the declaration of a singly linked linked-list node below.

```
typedef struct node {
  int value;
  __struct node*__ next; // pointer to the next element
} sll_node;
```

2) Let's convert the linked list to an array. Fill in the missing code.

```
int * to_array(sll_node *sll, int size) {
  int i = 0;
  int *arr = __malloc(size * sizeof(int))___;
  while (sll) {
    arr[i] = ___sll->value___________________;
    sll    = ___sll->next____________________;
    _______i++_______________;
  }
  return arr;
}
```

3) Finally, complete `delete_even()` that will delete every second element of the list. For example, given the lists below:

Before: $\boxed{\text{Node 1}} \rightarrow \boxed{\text{Node 2}} \rightarrow \boxed{\text{Node 3}} \rightarrow \boxed{\text{Node 4}}$

After: $\boxed{\text{Node 1}} \rightarrow \boxed{\text{Node 3}}$

Calling `delete_even()` on the list labeled "`Before`" will change it into the list labeled "`After`". All list nodes were created via dynamic memory allocation.

```
void delete_even(sll_node *sll) {
  sll_node *temp;
  if (!sll || !sll->next) return;
  temp = ___sll->next____;
  sll->next = ___temp->next (or sll->next->next)_______;
  free(____temp_______);
  delete_even(____sll->next______);
}
```

SID: ________________

Q5: MIPS with FUNctions (18 points)

The function **`countChars(char *str, char *target)`** returns the number of times characters in **`target`** appear in **`str`**. For example:

```
countChars("abc abc abc", "a") = 3
countChars("abc abc abc", "ab") = 6
countChars("abc abc abc", "abcd") = 9
```

```
int countChars(char *str, char *target) {
  int count = 0;
  while (*str) {
    count += isCharInStr(target, *str);
    str++;
  }
  return count;
}
```

The C code for countChars is given to you in the box on right. The helper function **`isCharInStr(char *target, char c)`** returns 1 if **`c`** is present in **`target`** and 0 if not.

Finish the implement of countChars in TAL MIPS below. You may not need every blank.

```
countChars:
     addiu $sp, $sp, ___-16____
     _sw $ra, 0($sp)___________   # Store onto the stack if needed
     _sw $s0, 4($sp)___________
     _sw $s1, 8($sp)___________
     _sw $s2, 12($sp)__________
     __________________________
     addiu $s0, $zero, 0           # We'll store the count in $s0
     addiu $s1, $a0, 0
     addiu $s2, $a1, 0
loop:
     addiu $a0, $s2, 0
     _lb $a1, 0($s1)___________
     beq _$a1, $zero, done_____
     jal isCharInStr
     _addu $s0, $s0, $v0_______
     _addiu $s1, $s1, 1________
     _j loop___________________
done:
     _addiu $v0, $s0, 0________   # Load from the stack if needed
     _lw $ra, 0($sp)___________
     _lw $s0, 4($sp)___________
     _lw $s1, 8($sp)___________
     _lw $s2, 12($sp)__________
     __________________________
     addiu $sp, $sp, ___16_____
     jr $ra
```

SID: ________________

Q6: MIPS Instruction Formats (16 points)

Convert the following TAL MIPS instructions into their machine code representation (binary format) or vice versa. For rows where you convert instructions to machine code, we've provided boxes to the right that you should fill in with the appropriate fields (in binary):

```
                MIPS                                  Machine Code
foo_bar:
      _______________________          0b00000000100000000001000000100001
loop: beq $a1 $0 end
```


```
      _______________________          0b00000000000000100001000001000000
      j loop                           0b00001000000000000000000000000001
end:  jr $ra
```


Q7: MIPS Addressing Modes (16 points)

We have a function that, when given a branch instruction, returns the number of bytes that the Program Counter (PC) would change by, i.e. **(PC_of_branch_target – PC_of_branch_instruction)**.

```
branchAmount(branch_inst):
  calculate the instruction offset from branch_inst
  convert the offset to byte addressing
  return PC_of_branch_target – PC_of_branch_instruction
```

Write `branchAmount` in TAL MIPS (no pseudoinstructions) .You may not need all the blanks. Assume that register **`$a0`** contains a valid branch instruction.

```
branchAmount:
     andi $t0, $a0, 0x8000        # Mask out a certain bit
     bne _$t0_, _$zero_, label1
     __andi $v0, $a0, 0xFFFF___
     __________________________
     j label2
label1:
     __lui $t1, 0xFFFF_________
     __________________________
     or $v0, $a0, $t1
label2:
     sll __$v0__, _$v0__, __2__   # Convert to byte addressing
     __addiu $v0, $v0, 4_______
label3:
     jr $ra
```