

CS/ECE 354 - Midterm Exam

Spring 2016

Name: _____

UW Student ID Number: _____

Wisc email id: _____

Instructions:

1. Do not open this exam until you are instructed to by the examiners.
2. Fill in your **NAME** and **UW Student ID** on all the pages of this exam.
3. You have **2 hours** to answer all the questions. This does not apply for students with special accommodations.
4. An **x86 cheat sheet** has been provided for your use during the exam on page number 18.
5. You are allowed to use **one 8.5 x 11 inch cheat sheet** (in addition to the x86 cheat sheet) during the exam. You **should submit your cheat sheet** along with your exam.
6. Please check that this exam contains **20 printed pages** with a total of **18 questions**.
7. All the x86 instructions in this exam follow the **AT&T syntax** discussed in the class lectures. Your answers must also follow the AT&T syntax.
8. The **last 2 pages (19 - 20)** of this exam are intentionally empty and you can use them for your **rough work** and other calculations. You can remove this blank sheet from the bundle for your calculations but **you should submit it along with your exam**.
9. If you need extra blank sheets for your rough work, you can ask the TAs proctoring your exam.
10. You are **NOT allowed** to use any electronic devices (including calculators) during the exam.

Part I - C Programming

C Basics

1. [2 points] In C, arguments to a function are always
 - a. Passed by value (i.e. the called function is given the values of its arguments in temporary variables rather than the originals)
 - b. Passed by reference (i.e. the called function has access to the original argument, not a local copy)
 - c. Non-pointer variables are passed by value and pointers are passed by reference

2. [3 points] What is the problem with the following code and how will you correct it?

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int *p = malloc(sizeof(int));
    *p = 42;
    p = malloc(sizeof(int));
    free(p);
}
```

ANSWER:

Pointers and Functions

3. [5 points] Does the following code run successfully to return 0 or does it generate a segmentation fault? If it runs fine, then what is the output? Otherwise explain why it segfaults.

A **segmentation fault** occurs when a program attempts to access a memory location that it is not allowed to access.

Note: The following code compiles without any errors!

```
#include <stdio.h>
#include <stdlib.h>

void populate(int *a)
{
    int *parray = malloc(2 * sizeof(int));
    parray[0] = 37;
    parray[1] = 73;
    a = parray;
}

int main()
{
    int *a = NULL;
    populate(a);
    printf("a[0] = %d and a[1] = %d\n", a[0], a[1]);
    return 0;
}
```

ANSWER:

Predict the output

Consider the following two programs, both of which are compiled and run on a CSL lab machine using the -m32 option.

```
sizeof(int) = 4 bytes
sizeof(char *) = 4 bytes
sizeof(char) = 1 byte
```

4. [3 points] What is the output of the code below on a 32-bit **little endian** machine?

Note: The %x format specifier in printf prints the contents in **hexadecimal** notation.

```

#include <stdio.h>
int main() {
    int number = 288;
    char *ptr;
    ptr = (char *) &number;
    printf("%x",*ptr);
    return 0;
}

```

ANSWER: _____

5. [4 points] What is the output of the code below?

```

#include <stdio.h>

int main() {
    char string[] = "BADGERS";
    char *ptr = string;
    *ptr = *ptr + 2;
    ptr = ptr + 2;
    printf("%c", *ptr);
    ptr--;
    printf("%c", *ptr);
    ptr = string;
    printf("%c", *ptr);
    return 0;
}

```

ANSWER: _____

Linked Lists

6. [2 points] Assuming the following line of code is inside the main() function, in which part of memory is the pointer variable `parray` allocated and in which part of memory is the 10 element integer array allocated?

```
int *parray = malloc (sizeof (int) * 10);
```

The 4 types of program memory are: Code, Data, Stack and Heap.

ANSWER:

parray is allocated in _____ memory.

10 element integer array is allocated in _____ memory.

7. [5 points] What does the following `mystery_function()` do?

```
struct node {
    int data;
    struct node *next;
}

void mystery_function(struct node *head)
{
    struct node *temp = head;
    struct node *prev;

    if (temp == NULL) {
        printf("Linked List is empty.\n");
    } else {
        prev = temp;
        while (temp != NULL) {
            temp = temp->next;
            free(prev);
            prev = temp;
        }
    }
}
```

- a. Frees all the nodes in the linked list one by one starting at the beginning of the linked list.
- b. Frees all the nodes in the linked list one by one starting at the end of the linked list.
- c. Frees all the nodes in the linked list one by one starting at the beginning of the linked list except the first node.
- d. Frees all the nodes in the linked list one by one starting at the beginning of the linked list except the last node.

Structures

8. [6 points] What is the size (in bytes) for the following structures on a 32-bit machine installed with the Linux Operating System?

sizeof(int) = 4 bytes

sizeof(short) = 2 bytes

sizeof(char) = 1 byte

Structure	Size (in bytes)
<pre>struct foo { int d1; char c1; int d2; }</pre>	
<pre>struct foo { int d1; char c1; int d2; char c2; short s; };</pre>	
<pre>struct foo { int d1; int d2; char c1; char c2; short s; };</pre>	
<pre>struct foo { char c1; int d1; short s; int d2; char c2; };</pre>	

Part II - Data Representation

9. [12 points] Suppose that x and y have byte values $0x93$ and $0x3F$, respectively. Fill in the following table indicating the byte values of the different C expressions. All values must be written in hexadecimal notation.

Expression	Value
$x \& y$	
$\sim x \mid \sim y$	
$x \& !y$	
$!x \mid \mid !y$	
$x \&\& \sim y$	
$x \ll 3$	
$x \gg 2$ (arithmetic)	

10. [8 points] The following function has a bug and doesn't work as expected. What is the issue with this function and how will you fix it?

```
// If x is greater than y, this function should return 1.
// Else, this function returns 0.
int is_greater(unsigned int x, unsigned int y)
{
    if (x - y > 0)
        return 1;
    else
        return 0;
}
```

ANSWER:

Part III - Assembly Programming

Addressing

11. [4 points] Assume the following memory layout, with the following values for the registers:

`%edx = 0x8049000`

`%ecx = 0x5`

Memory Address	Value in Memory
	...
0x8049024	0x8049028
0x8049020	0x8049024
0x804901c	0x8049020
0x8049018	0x804901c
0x8049014	0x8049018
0x8049010	0x8049014
0x804900c	0x8049010
0x8049008	0x804900c
0x8049004	0x8049008
0x8049000	0x8049004
	...

What is the value in the register `%eax` after each of the following assembly instructions?

Assembly Instruction	Value in register <code>%eax</code>
<code>movl \$0x8049000, %eax</code>	
<code>movl 0x8049000, %eax</code>	
<code>movl (%edx), %eax</code>	

<code>leal 4(%edx), %eax</code>	
<code>movl 4(%edx, %ecx, 4), %eax</code>	

Assembly to C

12. [6 points] Assume variables `a` and `b` are stored at `-0x8(%ebp)` and `-0x4(%ebp)` respectively. Write the equivalent C expressions for the following assembly snippets. The first one is already answered for you! :)

S.No.	Assembly Instruction	Corresponding C code
1	<pre>movl -0x4(%ebp), %eax movl (%eax), %eax movl %eax, -0x8(%ebp)</pre>	<code>a = *b;</code>
2	<pre>leal -0x4(%ebp), %eax movl %eax, -0x8(%ebp)</pre>	
3	<pre>movl -0x4(%ebp), %eax movl %eax, -0x8(%ebp)</pre>	
4	<pre>movl -0x4(%ebp), %eax movl (%eax), %edx movl -0x8(%ebp), %eax movl %edx, (%eax)</pre>	
5	<pre>movl -0x4(%ebp), %eax movl (%eax), %edx movl -0x8(%ebp), %eax movl (%eax), %eax addl %eax, %edx movl -0x4(%ebp), %eax movl %edx, (%eax)</pre>	

Arrays and Structures

13. [7 points] Assume variable **a** is stored starting at memory address **0x8049000**. What is the value of the memory location stored in the pointer variable **p** for the following cases? Write the value of **p** in hexadecimal notation.

sizeof(int) = 4 bytes

S.No.	C code	Hex value in pointer p
1	<pre>int a[10]; int *p = &a[7];</pre>	
2	<pre>struct foo { int x; int y; int z; }; struct foo a; int *p = &a.y;</pre>	
3	<pre>struct foo { int x[4]; int y[4]; }; struct foo a; int *p = &a.y[2];</pre>	
4	<pre>struct foo { int x[4]; int y[4]; }; struct foo a[10]; int *p = &a[7].y[2];</pre>	

Control Flags

14. [9 points] The function `fun()` takes three integer arguments `x`, `y`, and `z` and returns a character as shown below:

```
char fun(int x, int y, int z);
```

The arguments `x`, `y`, and `z` are stored at memory address `0x8(%ebp)`, `0xC(%ebp)`, and `0x10(%ebp)` respectively.

For the various assembly programs shown below, you are expected to fill in the correct data type cast (e.g. `char`, `short`, `int`) within the parenthesis () and the correct comparison operators (e.g. `<`, `<=`, `>`, `>=`, `!=`) in the blank space provided between the two operands. The first one is already answered for you! :)

The following are the data types and the comparison operators that you are allowed to use in the questions below:

Allowed data types: `int`, `char`, `short`, and `unsigned`

Allowed comparison operators: `>`, `<`, `>=`, `<=`, `!=`, `==`

`movsbl S,D` : Move sign-extended byte to double word ($D \leftarrow \text{SignExtend}(S)$)

`movsbw S,D` : Move sign-extended word to double word ($D \leftarrow \text{SignExtend}(S)$)

`cmpw S2,S1` : Compare word (based on `S1-S2`)

For the code snippets 1, 2, 3, and 4 (shown below), the char variable `t` is present at memory locations `-0x4(%ebp)`, `-0x6(%ebp)`, `-0x5(%ebp)`, and `-0x3(%ebp)` respectively.

S.No.	Assembly Instruction	Corresponding C code
1	<pre>movl 0x8(%ebp),%eax movsbl %al,%edx movl 0xc(%ebp),%eax cmpl %eax,%edx seta %al movb %al,-0x4(%ebp)</pre>	<pre>char t = (char)x > (unsigned)y;</pre>

2	<pre> movl 0x10(%ebp),%edx movl 0x8(%ebp),%eax cmpl %eax,%edx setbe %al movb %al,-0x6(%ebp) </pre>	<pre> char t = ()z ____ ()x; </pre>
3	<pre> movl 0x10(%ebp),%eax cmpl 0x8(%ebp),%eax setge %al movb %al,-0x5(%ebp) </pre>	<pre> char t = ()z ____ ()x; </pre>
4	<pre> movl 0x8(%ebp),%eax movsbw %al,%dx movl 0x10(%ebp),%eax cmpw %ax,%dx setne %al movb %al,-0x3(%ebp) </pre>	<pre> char t = ()x ____ ()z; </pre>

Jumps

15. [4 points] In the disassembled version of the machine code shown below, instructions at addresses 0x66, 0x6c, and 0x73 use relative encoding to encode jump targets. All the values in the disassembled code below (including the address column) are represented using hexadecimal numbers.

Address	Opcode	Assembly Code
0x62:	83 7d 08 00	cmpl \$0x0,0x8(%ebp)
0x66:	74 0d	je X
0x68:	83 7d 0c 00	cmpl \$0x0,0xc(%ebp)
0x6c:	74 07	je Y
0x6e:	b8 01 00 00 00	mov \$0x1,%eax
0x73:	eb 05	jmp Z
0x75:	b8 00 00 00 00	mov \$0x0,%eax
0x7a:	85 c0	test %eax,%eax
0x7c:	0f 9e c0	setle %al
0x7f:	88 45 ff	mov %al,-0x1(%ebp)

What is the value of the addresses X, Y and Z? The values of X, Y, and Z should be written in hexadecimal notation.

Remember the addresses X, Y, and Z, in assembly code will refer to actual addresses (e.g. 0x75) even though relative encoding is used in the machine code (opcode).

ANSWER:

X = 0x_____ Y = 0x_____ Z = 0x_____

Loops

16. [6 points] An assembly code and its corresponding C code is given below. Fill in the parts in the C code that are missing.

Assembly Code	C Code
<pre> loop_func: pushl %ebp movl %esp, %ebp subl \$16, %esp movl \$0, -4(%ebp) jmp .L2 .L3: movl 8(%ebp), %eax addl %eax, -4(%ebp) subl \$1, 8(%ebp) .L2: cmpl \$0, 8(%ebp) jg .L3 movl -4(%ebp), %eax leave ret </pre>	<pre> int loop_func(int n) { int sum = 0; while (_____) { _____; _____; } return sum; } </pre>

Functions

17. [8 points] Match the following C functions (C1, C2, C3 and C4) with their corresponding assembly functions (A1, A2, A3, and A4). Write your answers in the spaces provided at the end of the questions.

C program	Assembly Program
<p style="text-align: center;">C1</p> <pre>int func(int x, int y) { int result = x && y; return result; }</pre>	<p style="text-align: center;">A1</p> <pre>func: pushl %ebp movl %esp, %ebp subl \$16, %esp movl 8(%ebp), %eax orl 12(%ebp), %eax movl %eax, -4(%ebp) movl -4(%ebp), %eax leave ret</pre>
<p style="text-align: center;">C2</p> <pre>int func(int x, int y) { int result = x y; return result; }</pre>	<p style="text-align: center;">A2</p> <pre>func: pushl %ebp movl %esp, %ebp subl \$16, %esp cmpl \$0, 8(%ebp) je .L2 cmpl \$0, 12(%ebp) je .L2 movl \$1, %eax jmp .L3 .L2: movl \$0, %eax .L3: movl %eax, -4(%ebp) movl -4(%ebp), %eax leave ret</pre>

C3	A3
<pre>int func(int x, int y) { int result = x & y; return result; }</pre>	<pre>func: pushl %ebp movl %esp, %ebp subl \$16, %esp movl 8(%ebp), %eax andl 12(%ebp), %eax movl %eax, -4(%ebp) movl -4(%ebp), %eax leave ret</pre>
C4	A4
<pre>int func(int x, int y) { int result = x y; return result; }</pre>	<pre>func: pushl %ebp movl %esp, %ebp subl \$16, %esp cmpl \$0, 8(%ebp) jne .L2 cmpl \$0, 12(%ebp) je .L3 .L2: movl \$1, %eax jmp .L4 .L3: movl \$0, %eax .L4: movl %eax, -4(%ebp) movl -4(%ebp), %eax leave ret</pre>

Write your answers below: (If C1 matches with A4, write A4 in the space next to C1)

C1 - _____

C2 - _____

C3 - _____

C4 - _____

Recursion (Dream within a dream!)

18. [6 points] Consider the following recursive factorial function (shown in the lecture) in C and Assembly language. The line numbers for rfact in assembly are in decimal.

```
int rfact(int n)
{
    int result;
    if (n <= 1)
        result = 1;
    else
        result = n * rfact(n-1);
    return result;
}
```

Line#	Assembly Code
1.	rfact:
2.	pushl %ebp
3.	movl %esp, %ebp
4.	pushl %ebx
5.	subl \$4, %esp
6.	movl 8(%ebp), %ebx
7.	movl \$1, %eax
8.	cmpl \$1, %ebx
9.	jle .L53
10.	leal -1(%ebx), %eax
11.	movl %eax, (%esp)
12.	call rfact
13.	imull %ebx, %eax
14.	.L53:
15.	addl \$4, %esp
16.	popl %ebx
17.	popl %ebp
18.	ret

Questions:

1. Why do we push the %ebx register's value on the stack frame of rfact?
(Refer: Line# 4 in assembly code - `pushl %ebx`)

2. What is the purpose of the following 2 statements?

a. `subl $4, %esp` (Line number 5)

b. `addl $4, %esp` (Line number 15)

3. For every invocation of the function `rfact()` which register is used to store the value of its input argument?

4. What is the purpose of the following line of assembly code (Line number 10)?

```
leal -1(%ebx), %eax
```

5. Why are the following 2 lines (Line numbers 2 - 3) needed in `rfact()` function?

```
pushl %ebp
```

```
movl %esp, %ebp
```