CS 216 Exam 2 – Spring 2005 - SOLUTION

Name _______________________  Section _______________________
Email Address _________________  Student ID # _________________

Pledge:

This exam is closed note, closed book, with the exception of the “IBCM Principles of Operation” and the “Code Examples”, and the “Tiny Guide to x86 Assembly”. You may view these on-line - at the computer in front of you off of the cs216 home page – you may NOT use printed copies you may have brought with you. You may NOT refer to any other notes/books/slides/old exams etc.

You will have an hour and fifty minutes total to complete the exam. You may use calculators if needed (including the calculator on Windows).

It is an Honor Code violation to discuss this exam with ANYONE (including other students who have already taken the exam) until after 9:30pm Tuesday, April 5, 2005.

Good Luck!

<table>
<thead>
<tr>
<th></th>
<th>MAX</th>
<th>SCORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTAL</td>
<td>80</td>
<td></td>
</tr>
</tbody>
</table>
1. (4 points): Modify the picture above to show what happens to the stack and the contents of the registers after the following instructions have been executed. Assume all values are decimal.

```
pop eax
push ecx
mov [edx + 4], ebx
sub esp, 8
```

Modify the picture above to show what happens to the stack and the contents of the registers after the following instructions have been executed. Assume all values are decimal.
2. (3 points) Draw the binary search tree created by inserting these values in this order:

```
6  8  7  2  4  1  9  5  0
```

3. (2 points) Give a pre-order traversal of your tree shown above:

```
6  2  1  0  4  5  8  7  9
```

4. (2 points) Give a post-order traversal of your tree shown above:

```
0  1  5  4  2  7  9  8  6
```

5. (3 points) Delete the root of the tree shown above using one of the methods described in class. Draw the new tree here:

```
```

```
```
6. (3 points) Given the following tree:

```
     37
    /   \
   22    77
  /     /  \
 17    65    95
 /     /  \
5     30    50
```

Is it an AVL tree?  **YES**  or   **NO**  (If NO, circle the node(s) where the AVL property is violated.)

Why or why not (must answer for any credit)?

It is a BST, and at each node the height of its left and right subtrees differ by at most one.

7. (3 points) Given the following tree:

```
     67
    /   \
   20    70
  /     /  \
 42    68    89
 /     /  \
23    78    98
```

Is it an AVL tree?  **YES**  or   **NO**  (If NO, circle the node(s) where the AVL property is violated.)

Why or why not (must answer for any credit)?

At both node 20 and node 70, the height of the right subtree is 2 more than the height of the left subtree (differs by more than one).
8. (3 points) Given the following AVL tree:

![AVL Tree Diagram]

Insert the value 38 into the AVL tree above, doing any necessary rotations to maintain the AVL property.

(no rotations necessary)
9. (3 points) Given the following AVL tree:

```
  6
 /\   /
 3 10 4
 /\      /
1  8     9
```

Insert the value 9 into the AVL tree above, doing any necessary rotations to maintain the AVL property.

```
  6
 /\   /
 3 10 4
 /\      /
1  8     9
```

(Double Rotation)
10. (14 points total) Given the following two C++ functions:

```cpp
int important (int param1, int param2, int *param3) {
    int fancy;
    int simple;
    int boring;
    // Code (not shown) that does something useful and
    // puts useful values into ESI, EBX, and EDX.
    . . .
    fancy = param1 + param2;
    simple = fabulous (fancy, *param3);
    . . .
    // Code (not shown) that uses values of registers
    // ESI, EBX, and EDX from before the call to fabulous.
    return simple;
}

int fabulous(int super, int cool) {
    int bogus[3];
    return bogus[1];
}
```

Assume we have these global variables:

```cpp
int result;
int something;
```

Assume we have the following code in the main program:

```cpp
int main() {
    . . .
    // Code (not shown) that puts useful values into
    // registers ECX, EBX, and EDI.
    result = important (25, 13, &something);
    // Code (not shown) that uses the values in registers
    // ECX, EBX and EDI from before the call to important.
}
```
a) (7 points) Assuming that the compiler generates code using the C calling convention as discussed in class, draw a picture of the stack as it would appear immediately after the callee’s prolog in important has been executed. (location indicated approximately by the a)--→)
Be sure to include any values that would be pushed on to the stack by the caller’s (main’s) prolog (after start--→). For all of this question, assume that only the registers listed in the comments in each function and in main are used, and that only the registers that had to be saved were saved. Be sure to indicate what the contents of esp and ebp are by clearly drawing arrows from the boxes to the right to the stack.

b) (7 points) Draw a picture of the stack as it would appear immediately after the callee’s prolog in fabulous has been executed. (location indicated approximately by the b)--→). (If you want, you can draw only the additional things pushed on the stack above what is already there in question a).) Be sure to indicate what the contents of esp and ebp are.
11. (25 points) Please write the x86 code that implements a recursive function sum_array as shown below. sum_array will return the sum of all values in the array using recursion rather than iteration. The integer size indicates how many elements are in the input array. You can assume that the array input contains only non-negative integer values. You can also assume that size >= 1.

Detailed comments will help us assign you partial credit in case your solution is not perfect, but are not required for full credit. You should follow the C++ calling convention as described in class. You should allocate space for the local variables shown below regardless of whether you use them in your code or not. Feel free to allocate more local variables if needed. Avoid saving registers whenever possible. Writing pseudo-code at first may be very helpful!! Remember that arrays are passed by reference in C++. Be sure to include the prolog and the epilog. You should refer to parameters and local variables as we have done in class and in the x86 handout.

Here are a few example calls in C++ and what the results should be:

When: my_array = {12, 0, 20, 6, 4}, sum_array(my_array, 5) returns 42
When: my_array = {1, 5, 6, 5}, sum_array(my_array, 4) returns 17

**PLEASE NOTE THAT YOUR FUNCTION MUST BE RECURSIVE (i.e. include a call to sum_array) FOR FULL CREDIT!! Your function should follow the calling convention and NOT try to do any optimizations. A maximum of 10 points are possible if your function is not recursive.

```c
int sum_array (int input[], int size) {
    int temp;
    int i;

    // write prolog of sum_array here in x86:
    // (remember to write the epilog at the end)
```
// (continue your code on this page if needed)

sum_array PROC
prolog:
push ebp
mov ebp, esp
sub esp, 8

mov ecx, [ebp+8] ; ecx = &input[0]
cmp DWORD PTR [ebp+12], 1 ; if size > 1,
jg else_clause ; go to else clause

mov eax, [ecx] ; return input[0]
jmp epilog

else_clause:
push ecx ; need ecx after call
; or could read from [ebp+8] again

sub DWORD PTR [ebp+12], 1 ; push two parameters
push [ebp+12] ; push size - 1
mov ebx, [ebp+8]
add ebx, 4
push ebx ; push &input[1]
call sum_array

add esp, 8 ; remove params
pop ecx ; restore ecx
add eax, [ecx] ; eax = return value + input[0]

epilog:
mov esp, ebp
pop ebp
ret

sum_array ENDP
END
In case it is useful to you, the following table provides the operation code definitions for IBCM. It is the same as the table in class handouts.

<table>
<thead>
<tr>
<th>OP</th>
<th>Mnem</th>
<th>Note</th>
<th>OP</th>
<th>Mnem</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>halt</td>
<td>halt!</td>
<td>8</td>
<td>or</td>
<td>logical OR mem into accum</td>
</tr>
<tr>
<td>1</td>
<td>io</td>
<td>bit 4 I/O, bit 5 hex/ascii</td>
<td>9</td>
<td>xor</td>
<td>logical XOR mem into accum</td>
</tr>
<tr>
<td>2</td>
<td>shift</td>
<td>bit 4 shift/rotate, bit 5 left/right</td>
<td>A</td>
<td>not</td>
<td>logical complement of accum</td>
</tr>
<tr>
<td>3</td>
<td>load</td>
<td>load accum from mem</td>
<td>B</td>
<td>nop</td>
<td>no operation, do nothing</td>
</tr>
<tr>
<td>4</td>
<td>store</td>
<td>store accum to mem</td>
<td>C</td>
<td>jmp</td>
<td>unconditional jump</td>
</tr>
<tr>
<td>5</td>
<td>add</td>
<td>add mem to accum</td>
<td>D</td>
<td>jmpe</td>
<td>jump to addr if accum is 0</td>
</tr>
<tr>
<td>6</td>
<td>sub</td>
<td>subtract mem from accum</td>
<td>E</td>
<td>jmpl</td>
<td>jump to addr if accum &lt; 0</td>
</tr>
<tr>
<td>7</td>
<td>and</td>
<td>logical AND mem into accum</td>
<td>F</td>
<td>brl</td>
<td>jump to addr; set accum to value of PC just before jump</td>
</tr>
</tbody>
</table>

12. (15 points) Write an IBCM program that implements the following pseudocode. Your answer should be machine code (encoded). First write symbolic IBCM instructions for significant partial credit. Then encode the IBCM instructions for full credit. For full credit on this question, encoded instructions and symbolic assembly is what is required. Comments will help us assign you partial credit in case your solution is not perfect, but are not required for full credit. You can assume that x and y will be integers $\geq 0$. Please clearly indicate your final answer.

```ibcm
read x;
if (x > 5) {
    read y;
    print y + x;
} else {
    print x;
}
halt
```

Write your IBCM code on the next page....
<table>
<thead>
<tr>
<th>mem</th>
<th>loc</th>
<th>label</th>
<th>op</th>
<th>addr</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>C005</td>
<td>00</td>
<td>jmp</td>
<td>start</td>
<td></td>
<td>go to start</td>
</tr>
<tr>
<td>0000</td>
<td>01</td>
<td>x</td>
<td>DW</td>
<td>0</td>
<td>variable x</td>
</tr>
<tr>
<td>0000</td>
<td>02</td>
<td>y</td>
<td>DW</td>
<td>0</td>
<td>variable y</td>
</tr>
<tr>
<td>0005</td>
<td>03</td>
<td>five</td>
<td>DW</td>
<td></td>
<td>constant 5</td>
</tr>
<tr>
<td>0000</td>
<td>04</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>05</td>
<td>start</td>
<td>readH</td>
<td></td>
<td>read in x</td>
</tr>
<tr>
<td>4001</td>
<td>06</td>
<td>store</td>
<td>x</td>
<td></td>
<td>store x</td>
</tr>
<tr>
<td>6003</td>
<td>07</td>
<td>sub</td>
<td>five</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E00E</td>
<td>08</td>
<td>jmpl</td>
<td>else</td>
<td></td>
<td>if x &lt; 5 goto else</td>
</tr>
<tr>
<td>D00E</td>
<td>09</td>
<td>jmpe</td>
<td>else</td>
<td></td>
<td>if x == 5 goto else</td>
</tr>
<tr>
<td>1000</td>
<td>0A</td>
<td>readH</td>
<td>y</td>
<td></td>
<td>read in y</td>
</tr>
<tr>
<td>5001</td>
<td>0B</td>
<td>add</td>
<td>x</td>
<td></td>
<td>accum = x + y</td>
</tr>
<tr>
<td>1800</td>
<td>0C</td>
<td>printH</td>
<td></td>
<td></td>
<td>print x + y</td>
</tr>
<tr>
<td>C010</td>
<td>0D</td>
<td>jmp</td>
<td>xit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3001</td>
<td>0E</td>
<td>else</td>
<td>load</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>1800</td>
<td>0F</td>
<td>printH</td>
<td></td>
<td></td>
<td>print x</td>
</tr>
<tr>
<td>0000</td>
<td>10</td>
<td>xit</td>
<td>halt</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>