SOLUTION FOR

CS 216 Exam 2 – Fall 2005

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 forty-five 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 noon, Thursday, Nov. 17, 2005.

Good Luck!

<table>
<thead>
<tr>
<th></th>
<th>MAX</th>
<th>SCORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTAL</td>
<td>72</td>
<td>74</td>
</tr>
</tbody>
</table>
1. (6 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
mov ebx, [edx + 12]
mov [edx], ebp
sub ebp, 4
xor ecx, ecx
```

Memory Address

<table>
<thead>
<tr>
<th>eax</th>
<th>996</th>
<th>92</th>
</tr>
</thead>
<tbody>
<tr>
<td>ebx</td>
<td>992</td>
<td>00</td>
</tr>
<tr>
<td>ecx</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>edx</td>
<td>988</td>
<td></td>
</tr>
<tr>
<td>esp</td>
<td>1004</td>
<td></td>
</tr>
<tr>
<td>ebp</td>
<td>1012</td>
<td>03</td>
</tr>
</tbody>
</table>

2. (6 points) In the worst case, how long does it take to determine that an element is NOT in a hash table. Assume that the hash table has been implemented using separate chaining, where each bucket is a linked-list. You must describe how you came up with your answer for any credit. Also, what would the worst-case be if each bucket is an AVL tree?

- A. \( \Theta(n) \) when \( n \) is number of items in table
- B. \( \Theta(1) \) if all items hash to same bucket
- C. \( \Theta(\log n) \)

3. (4 points) Give the Big-Oh for the average case time-complexity for the find operation for an AVL tree. What is it for a hash-table using separate chaining?

- \( \Theta(\log n) \)
- \( \Theta(1) \) or \( \Theta(\lambda) \) (the former assumes \( \lambda \) value that's good)

4. (2 points) What load-factor does the textbook recommend for a hash-table when we use separate chaining? When we use open-addressing?

- \( \lambda = 1 \)
- \( \lambda = 0.5 \)
5. (6 points) Draw the contents of the hash table in the boxes below given the following conditions. Note that the calculator in Windows will do modulo arithmetic.
Choose Start->run, and type calc. Select view->scientific.
No partial credit – check your answers carefully!

The size of the hash table is 7.
Use separate chaining, where each bucket is an unordered linked list.
The hash function used is \( H(k) = k \mod \text{table size} \)

What values will be in the hash table after the following sequence of insertions (draw in boxes below):

\[ 16, 7, 32, 11, 46, 21 \]
6. (5 points) Draw the contents of the hash table in the boxes below given the following conditions. Note that the calculator in Windows will do modulo arithmetic.
Choose Start->run and type calc. Select view->scientific

No partial credit – check your answers carefully!
The size of the hash table is 8.
Open addressing and linear probing is used to resolve collisions.
The hash function used is $H(k) = (k^{*}2) \mod \text{tables size}$

What values will be in the hash table after the following sequence of insertions (draw in boxes below)?

7, 16, 22, 30, 10

\[
\begin{array}{c}
0 \\
1 \\
2 \\
3 \\
4 \\
5 \\
6 \\
7 \\
\end{array}
\begin{array}{c}
| \downarrow \\
| \downarrow \\
| \downarrow \\
| \downarrow \\
| \downarrow \\
| \downarrow \\
| \downarrow \\
\end{array}
\begin{array}{c}
7 \\
10 \\
\end{array}
\begin{array}{c}
1 \circ 2 \\
3 \circ 4 \\
5 \circ 6 \\
\end{array}
\]
7. (4 points) Draw the binary search tree created by inserting these values in this order:

\[ 7 \ 9 \ 11 \ 2 \ 8 \ 10 \ 5 \ 3 \]

8. (2 points) Give a pre-order traversal of your tree shown above:

\[ 7 \ 2 \ 5 \ 3 \ 9 \ 8 \ 11 \ 10 \]

9. (2 points) Give an in-order traversal of your tree shown above:

\[ 2 \ 3 \ 5 \ 7 \ 8 \ 9 \ 10 \ 11 \]

10. (4 points) Delete the root of the tree shown above using one of the methods described in class. Draw the new tree here:

\[ 8 \]

\[ 2 \]

\[ 5 \]

\[ 3 \]

\[ 10 \]

\[ 9 \]

\[ 11 \]
11. (3 points) Given the following tree:

```
   32
  /   \
24    44
 /     /\    \
7     29   60
       /   /\    \
      26  31  65
```

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)?

NO  depth of left sub-tree ad right sub-tree of (44) differ by more than 1

12. (4 points) Given the following AVL tree:

```
   5
  / \    
3   8
 / \   
2   6
```

Insert the value 7 into the AVL tree above, doing any necessary rotations to maintain the AVL property.

```
   5
  /   \
3     7
 /     /\    \
2     6   8
```
13. (10 pts) Write x86 assembly code that implements the following pseudo-code. You must comment your code.

```plaintext
x = 1;
y = 4;
while (y >= x) {
    y;
}
```

There are many ways to code this, but they all have much in common. We looked for the following things:

a) Declare variables with initial values
   --- or ---
   move values into memory locations
b) Have a label for the top of the loop
c) Compare and jump on the result
   Note: If you did cmp y, x you needed to use jl
        If you did cmp x, y you needed to use jg
d) Decrement the register containing y (or the memory location for y)
e) Jump back to the top of the loop
f) Save the register containing y back to y's memory location

We took off a point or two if you didn't store x and y in memory but only used registers.
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>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>halt</td>
<td>halt!</td>
</tr>
<tr>
<td>1</td>
<td>io</td>
<td>bit 4 I/O, bit 5 hex/ascii</td>
</tr>
<tr>
<td>2</td>
<td>shift</td>
<td>bit 4 shift/rotate, bit 5 left/right</td>
</tr>
<tr>
<td>3</td>
<td>load</td>
<td>load accum from mem</td>
</tr>
<tr>
<td>4</td>
<td>store</td>
<td>store accum to mem</td>
</tr>
<tr>
<td>5</td>
<td>add</td>
<td>add mem to accum</td>
</tr>
<tr>
<td>6</td>
<td>sub</td>
<td>subtract mem from accum</td>
</tr>
<tr>
<td>7</td>
<td>and</td>
<td>logical AND mem into accum</td>
</tr>
<tr>
<td>8</td>
<td>or</td>
<td>logical OR mem into accum</td>
</tr>
<tr>
<td>9</td>
<td>xor</td>
<td>logical XOR mem into accum</td>
</tr>
<tr>
<td>A</td>
<td>not</td>
<td>logical complement of accum</td>
</tr>
<tr>
<td>B</td>
<td>nop</td>
<td>no operation, do nothing</td>
</tr>
<tr>
<td>C</td>
<td>jmp</td>
<td>unconditional jump</td>
</tr>
<tr>
<td>D</td>
<td>jmpe</td>
<td>jump to addr if accum is 0</td>
</tr>
<tr>
<td>E</td>
<td>jimpl</td>
<td>jump to addr if accum &lt; 0</td>
</tr>
<tr>
<td>F</td>
<td>brl</td>
<td>jump to addr, set accum to value of PC just before jump</td>
</tr>
</tbody>
</table>

14. (16 points) Write an IBCM program that implements the following pseudo-code. 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 >= 0. Please clearly indicate your final answer.

Note: the pseudo-code below is very similar to that in the previous x86 question!

```plaintext
read x;
read y;
while (y >= x) {
    print y;
    --y;
}
halt
```

<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>c006</td>
<td>00</td>
<td></td>
<td>jmp</td>
<td>start</td>
<td>skip variables</td>
</tr>
<tr>
<td>0000</td>
<td>01</td>
<td>x</td>
<td>dw</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0000</td>
<td>02</td>
<td>y</td>
<td>dw</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>03</td>
<td>one</td>
<td>dw</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0000</td>
<td>04</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000</td>
<td>05</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>06</td>
<td>start</td>
<td>readH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4001</td>
<td>07</td>
<td>store</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>08</td>
<td></td>
<td>readH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4002</td>
<td>09</td>
<td>store</td>
<td>y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3002</td>
<td>0A</td>
<td>loop</td>
<td>load</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>6001</td>
<td>0B</td>
<td>sub</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E012</td>
<td>0C</td>
<td>jimpl</td>
<td>xit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3002</td>
<td>0D</td>
<td>load</td>
<td>y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1800</td>
<td>0E</td>
<td></td>
<td>printH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6003</td>
<td>0F</td>
<td>sub</td>
<td>one</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4002</td>
<td>10</td>
<td>store</td>
<td>y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C00A</td>
<td>11</td>
<td>jmp</td>
<td>loop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000</td>
<td>12</td>
<td>xit</td>
<td>halt</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exam 2