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).

Good Luck!

<table>
<thead>
<tr>
<th></th>
<th>MAX</th>
<th>SCORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTAL</td>
<td>71</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 ecx
mov esi, [ebx + 1000]
mov [2 * eax], ebp
```

2. (4 points) Given the following infix expression: 

\[
((a + b) / ((c - d) * (k + j))) + z
\]

a) Write it as a prefix expression:

\[
+ / + a b * - c d + k j z
\]

b) Write it as a postfix expression:

\[
a b + c d - k j + * / z +
\]
3. (3 points) Given the following AVL tree:

```
     8
    / \     \
   5   9
  /     /
3      16
```

Insert the value 17 into the AVL tree above, doing any necessary rotations to maintain the AVL property.

```
     8
    / \     \
   5   16
  /     /
3     9  17
```
4. (3 points) Given the following AVL tree:

```
    7
   / \
  4   10
 / \   |
3   5   6
```

Insert the value 6 into the AVL tree above, doing any necessary rotations to maintain the AVL property.

```
    6
   / |
  3   5
   \ |
   4 10
```

```
5. (3 points total) Describe the fetch execute cycle in pseudocode:

```plaintext
while (power is on) {
    IR = mem [PC]
    PC++
    Execute instruction in IR
}
```
6. (14 points total) Given the following two C++ functions:

```cpp
int meal_plan (int input_a, int input_b) {
  int apples;
  int oranges;

  a)------
  // code that does something useful (not shown)
  // puts useful values into ESI, ECX, and EDX
  apples = input_a - input_b;
  oranges = compare(apples, input_b + 13, 52);
  // code that uses values of registers ESI, ECX, and EDX.
  // from before the call to compare.
  // (not shown)
  return apples;
}

int compare(int param_1, int param_2, int param_3) {
  int temp[3];
  int answer;

  b)------
  // code that does something useful (not shown)
  // code that uses registers EDI and EBX (not shown)
  return answer;
}
```

Assume we have the following code in the main program:

```cpp
int final_result;
// code not shown that puts useful values in registers
// EBX and EDI.
start------
  final_result = meal_plan(100, 7);
  // code not shown that uses the values in registers
  // EBX and EDI from before the call to meal_plan.
```
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 `meal_plan 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.

```
Memory

Lower Addresses
- Main’s ESI
- apples
- oranges
- main’s EBP
- Return address
- 100
- 7
- main’s EBX

Higher Addresses
```

b) (7 points) Draw a picture of the stack as it would appear immediately after the callee’s prolog in `compare 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.

```
Memory

Lower Addresses
- meal_plan’s EDI
- temp[0]
- temp[1]
- temp[2]
- answer
- meal_plan’s EBP
- Return address
- apples
- input b + 13
- 52
- meal_plan’s EDX
- meal_plan’s ECX
- Stuff from a)

Higher Addresses
```

---

Exam 2  
CS216 – Anderson – Fall 2004  
Page 7 of 11
7. (25 points) Please write the x86 code that implements a function `remove_max` as shown below. `remove_max` will find and return the maximum value in the array. In addition, `remove_max` will modify the array to write a value of 0 in place of the removed value. 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 $\geq 1$. If more than one instance of the maximum value exists you should only remove the first instance.

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, 36, 14}, \hspace{1cm} \text{remove_max (my_array, 5)} \hspace{1cm} \text{returns 36 and modifies my_array to contain: } \{12, 0, 20, 0, 14\}

When: my_array = {1, 56, 10, 2, 56, 1, 24}, \hspace{1cm} \text{remove_max (my_array, 7)} \hspace{1cm} \text{returns 56 and modifies my_array to contain: } \{1, 0, 10, 2, 56, 1, 24\}

```c
int remove_max (int input[], int size) {
    int location;
    int i;

    (see next page for solution)
```
push ebp
mov ebp, esp ; allocate local variables
sub esp, 8
mov ebx, [ebp+8] ; ebx = &array
mov ecx, 0 ; ecx = 0 (loop counter i)
mov eax, 0 ; Initialize eax to contain the current max value
mov edx, 0 ; Initialize edx to contain the location of current max.

loop_start:
cmp ecx, [ebp+12] ; if i >= size, exit loop
jge done ; exit loop
cmp [ebx + 4*ecx], eax ; if array[i] > current max
jle not_greater
mov eax, [ebx + 4*ecx] ; save new current max
mov edx, ecx ; save location of new current max

not_greater:
inc ecx ; i++
jmp loop_start

done:
mov DWORD PTR [ebx + 4*edx], 0 ; put a zero in place of the max
; return value is already in eax

epilog:
mov esp, ebp
pop ebp
ret
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>

8. (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 \( x \) will be an integer \( \geq 0 \). Please clearly indicate your final answer.

```
read x;
read y;
while (x > y) {
    print x - 3;
    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>C006</td>
<td>00</td>
<td>start</td>
<td>jmp</td>
<td>0</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></td>
<td>variable y</td>
</tr>
<tr>
<td>0001</td>
<td>03</td>
<td>one</td>
<td>DW</td>
<td></td>
<td>constant 1</td>
</tr>
<tr>
<td>0003</td>
<td>04</td>
<td>three</td>
<td>DW</td>
<td></td>
<td>constant 3</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>read in x</td>
</tr>
<tr>
<td>4001</td>
<td>07</td>
<td>store</td>
<td>x</td>
<td></td>
<td>store x</td>
</tr>
<tr>
<td>1000</td>
<td>08</td>
<td>readH</td>
<td>y</td>
<td></td>
<td>read in y</td>
</tr>
<tr>
<td>4002</td>
<td>09</td>
<td>store</td>
<td>y</td>
<td></td>
<td>store y</td>
</tr>
<tr>
<td>3001</td>
<td>0A</td>
<td>loop</td>
<td>load</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>6002</td>
<td>0B</td>
<td>sub</td>
<td>y</td>
<td>x - y</td>
<td></td>
</tr>
<tr>
<td>E015</td>
<td>0C</td>
<td>jmpl</td>
<td>xit</td>
<td></td>
<td>If y &gt; x goto xit</td>
</tr>
<tr>
<td>D015</td>
<td>0D</td>
<td>jmpe</td>
<td>xit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3001</td>
<td>0E</td>
<td>load</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6004</td>
<td>0F</td>
<td>sub</td>
<td>three</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1800</td>
<td>10</td>
<td>printH</td>
<td>x</td>
<td></td>
<td>print x - 3</td>
</tr>
<tr>
<td>3001</td>
<td>11</td>
<td>load</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6003</td>
<td>12</td>
<td>sub</td>
<td>one</td>
<td>x--</td>
<td></td>
</tr>
<tr>
<td>4001</td>
<td>13</td>
<td>store</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C00A</td>
<td>14</td>
<td>jmp</td>
<td>loop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000</td>
<td>15</td>
<td>xit</td>
<td>halt</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>