Problem PA-2 (1 part)

Part A: The function `Bar` (below left) calls function `Foo` after completing code block 1. Write MIPS assembly code that properly calls `Foo`. Include all instructions between code block 1 and code block 2. Symbolically label all required stack entries and give their values if they are known (below right).

```mips
int Bar() {
    int V[] = {9, 4, 1};
    int R = V[2];

    (code block 1)
    R = Foo(R, &R, V)

    (code block 2)
}
```

### Activation Frames

<table>
<thead>
<tr>
<th>Bar’s FP</th>
<th>XXX</th>
<th>XXX</th>
</tr>
</thead>
<tbody>
<tr>
<td>9900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9896</td>
<td>V[2]</td>
<td>1</td>
</tr>
<tr>
<td>9892</td>
<td>V[1]</td>
<td>4</td>
</tr>
<tr>
<td>9888</td>
<td>V[0]</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RA</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>9880</td>
<td></td>
</tr>
<tr>
<td>9876</td>
<td>FP</td>
</tr>
<tr>
<td>9872</td>
<td>R</td>
</tr>
<tr>
<td>9868</td>
<td>&amp;R</td>
</tr>
<tr>
<td>9864</td>
<td>V</td>
</tr>
<tr>
<td>9860</td>
<td>RV</td>
</tr>
<tr>
<td>9856</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>label</th>
<th>instruction</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>addi $29, $29, -24</td>
<td># allocate activation frame</td>
<td></td>
</tr>
<tr>
<td>sw $31, 20($29)</td>
<td># preserve RA</td>
<td></td>
</tr>
<tr>
<td>sw $30, 16($29)</td>
<td># preserve FP</td>
<td></td>
</tr>
<tr>
<td>lw $1, -16($30)</td>
<td># copy R's value</td>
<td></td>
</tr>
<tr>
<td>sw $1, 12($29)</td>
<td># store in activation frame</td>
<td></td>
</tr>
<tr>
<td>addi $1, $30, -16</td>
<td># compute R's effective address</td>
<td></td>
</tr>
<tr>
<td>sw $1, 8($29)</td>
<td># store in activation frame</td>
<td></td>
</tr>
<tr>
<td>addi $1, $30, -12</td>
<td># compute V's base address</td>
<td></td>
</tr>
<tr>
<td>sw $1, 4($29)</td>
<td># store in activation frame</td>
<td></td>
</tr>
<tr>
<td>jal Foo</td>
<td># call Foo</td>
<td></td>
</tr>
<tr>
<td>lw $31, 20($29)</td>
<td># restore RA</td>
<td></td>
</tr>
<tr>
<td>lw $30, 16($29)</td>
<td># restore FP</td>
<td></td>
</tr>
<tr>
<td>lw $1, 0($29)</td>
<td># load return value</td>
<td></td>
</tr>
<tr>
<td>sw $1, -16($30)</td>
<td># store in R</td>
<td></td>
</tr>
<tr>
<td>addi $29, $29, 24</td>
<td># deallocate activation frame</td>
<td></td>
</tr>
</tbody>
</table>
Part B: Assume the code above calls a function `int Foo(int A, *B, C[])` that properly completes and references its activation frame. Write short code fragments for `Foo` that perform the access of the specified function parameters below. Assume `Foo`’s frame pointer is properly set and no parameters are present in registers (i.e., access all parameters from `Foo`’s activation frame).

$5 = *B$  
(dereference B and store the value in $5$)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw $5, 8($30)</td>
<td># get pointer B</td>
</tr>
<tr>
<td>lw $5, 0($5)</td>
<td># dereference B</td>
</tr>
</tbody>
</table>

$C[2] = \$6$  
(stores the contents of register 6 into C[2])

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw $1, 4($30)</td>
<td># load base of array C</td>
</tr>
<tr>
<td>sw $6, 8($1)</td>
<td># store $6 at C[2]</td>
</tr>
</tbody>
</table>

A--  
(decrements A)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw $1, 12($30)</td>
<td># load A</td>
</tr>
<tr>
<td>addi $1, $1, -1</td>
<td># decrement A</td>
</tr>
<tr>
<td>sw $1, 12($30)</td>
<td># store A</td>
</tr>
</tbody>
</table>

return A  
(returns A as the return value)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw $1, 12($30)</td>
<td># load A</td>
</tr>
<tr>
<td>sw $1, 0($30)</td>
<td># store in RV slot</td>
</tr>
<tr>
<td>add $29, $30, $0</td>
<td># deallocate activation frame</td>
</tr>
<tr>
<td>jr $31</td>
<td># return to caller</td>
</tr>
</tbody>
</table>