Proc2:  
```
addi $29, $29, -4  
sw $16, 0($29)  
muli $8, $5, 4  
add $8, $4, $8  
lw $16, 0($8)  
lw $9, 4($8)  
sw $9, 0($8)  
sw $16, 4($8)  
lw $16, 0($29)  
addi $29, $29, 4  
jr $31
```
Notes

Register 0 always has the value of 0.

A subroutine call is made using a jump-and-link instruction (jal). The return address is placed in register 31, not on the stack because there is no implicit stack pointer defined in the instruction set architecture.

For subroutines, the following conventions are used:

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Register Number</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero</td>
<td>0</td>
<td>Constant 0</td>
</tr>
<tr>
<td>at</td>
<td>1</td>
<td>Reserved for assembler</td>
</tr>
<tr>
<td>v0,v1</td>
<td>2,3</td>
<td>Function return results</td>
</tr>
<tr>
<td>a0-a3</td>
<td>4-7</td>
<td>Arguments 1-4</td>
</tr>
<tr>
<td>t0-t7</td>
<td>8-15</td>
<td>Temporary (not preserved across procedure call)</td>
</tr>
<tr>
<td>s0-s7</td>
<td>16-23</td>
<td>Saved temporary (preserved across procedure call)</td>
</tr>
<tr>
<td>t8,t9</td>
<td>24,25</td>
<td>Temporary (not preserved across procedure call)</td>
</tr>
<tr>
<td>k0,k1</td>
<td>26,27</td>
<td>Reserved for OS kernel</td>
</tr>
<tr>
<td>gp</td>
<td>28</td>
<td>Pointer to global area</td>
</tr>
<tr>
<td>sp</td>
<td>29</td>
<td>Stack pointer</td>
</tr>
<tr>
<td>fp</td>
<td>30</td>
<td>Frame pointer</td>
</tr>
<tr>
<td>ra</td>
<td>31</td>
<td>Return address (used by procedure call)</td>
</tr>
</tbody>
</table>

Remember that these are conventions set by the compiler writer. Except for register 31 being used to hold a return address when a jal instruction is executed, there are no side-effects or special meanings assigned to the registers in the instruction set definition.

The procedure call stack grows from larger addresses to small addresses.

```c
int v[10000];

sort (int v[], int n);
{
    int i, j;
    for (i = 0; i < n; i = i+1) {
        for (j = i-1; j>=0 && v[j]>v[j+1]; j = j-1) {
            swap(v,j);
        }
    }
}

swap (int v[], int k);
{
    int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```